BC

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 873 013 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
21.10.1998 Bulletin 1998/43

(51) Int. Cl.[6]: **H04N 7/00**, G06K 7/14

(21) Application number: **97119217.4**

(22) Date of filing: **04.11.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **05.11.1996 US 30360 P**

(71) Applicant: **WELCH ALLYN, INC.**
Skaneateles Falls New York 13153 (US)

(72) Inventors:
- **Parker, James A.**
  Liverpool, NY 13090 (US)
- **Ehrhart, Michael A.**
  Liverpool, NY 13090 (US)

(74) Representative:
Leineweber, Jürgen, Dipl.-Phys. et al
Aggerstrasse 24
50859 Köln (DE)

(54) Decoding of real time video imaging

(57) A process allows an image capturing apparatus to be integrated with a personal computer to continuously display a video image of the imaging apparatus. Upon proper input by a user, or automatically after a timed interval, a snapshot of the video image is captured. An autodiscrimination process of the captured video image automatically decodes any bar-coded information present in the captured image and outputs the information.

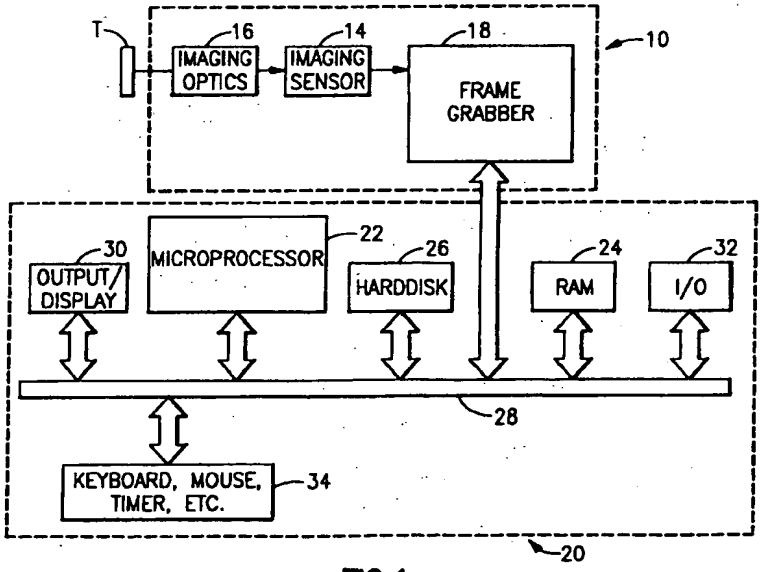


FIG.1

EP 0 873 013 A2

Description

FIELD OF THE INVENTION

This invention relates to image capturing apparatus, and more particularly to a method of capturing and decoding bar code information in real time from a continuously displayed video signal of a particular target.

BACKGROUND OF THE INVENTION

Image capture devices are known in the prior art for allowing diagnostic inspections to be performed, such as for surgical or other medical procedures with minimum human intervention. Such devices include video output for allowing real time images of a target of interest to be viewed. Examples of such devices used for medical purposes are described in U.S. Patent Nos. 4,755,873, and 4,651,202, among others, which allow the image to be continuously viewed on a video monitor. Similar devices, such as borescopes, are used for inspection of steam vessels, automotive engines, and other applications extending into the military, industrial and scientific fields.

In addition, bar code readers are also known for reading 1D and 2D bar code symbols, such as bar coded information in supermarkets, etc. A variety of different bar code symbols are now known, for the 1D bar code symbologies a number of them have been developed to allow the encoding of larger amounts of data, including Code 49, as described in U.S. Patent No. 4,794,239, issued to Allais, and PDF 417, as described in U.S. Patent No. 5,340,786, issued to Pavlidus, et al. In these patents, stacked symbols partition the encoded data into multiple rows, each including a respective 1D bar code pattern, all or most of which must be scanned and decoded, then linked together to form a complete message. Two dimensional (2D) matrix symbologies, have also been developed which offer orientation-free scanning and greater data densities and capacities than their 1D counterparts. 2D matrix codes encode data as dark or light data elements within a regular polygonal matrix, accompanied by graphical finder, orientation and reference structures.

Bar code readers are known which discriminate between the different types of symbologies of each of the above 1D and 2D types. For example, optical readers capable of 1D autodiscrimination are well known in the art. An early example of such a reader is the SCANTEAM™ 3000, manufactured by Welch Allyn, Inc.

Optical readers which are capable of 1D/2D discrimination are less well known in the art, in that 2D symbologies are relatively new innovations. An example of a hand-held reader having such capability is described in copending, commonly assigned USSN 08/504,643. A stationary 2D image sensor is described in copending, commonly assigned USSN 08/516,185, each of which is hereby incorporated by reference in their entirety. Most recently, a barcode reader which performs 1D/2D autodiscrimination of a target having multiple symbols is described in copending and commonly assigned U.S. SN 08/697,914, filed September 3, 1996. The reader captures a field of view and autodiscriminates between the symbols within the captured field of view.

Typically, if a video image obtained with an image sensor and displayed onto a computer monitor contains barcoded information, the following occurs: First, the image is captured into a named file saved by the computer, converted to disk or otherwise. The user then must separately load a bar code decoding program such as the 1D/2D program, described above, into the system and load the disk separately as a file for execution by the bar decoding program.

There are a number of problems with this approach. First, if the image is not properly resolved by the digital camera, or other imaging device, then the stored image can not be decoded properly. This means that the user must separately reaim the imaging device, download a new stored image, rename the file, reload the bar code decoding program, and reload the newly stored image as input into the decoding program. Several such iterations might be needed, with each iteration taking a considerable amount of time, and producing frustration and inconvenience for the user.

Furthermore, this situation exacerbates if a target of interest includes widely scattered symbols that are spaced by more than a single field of view, because multiple decoding operations would be required. As noted, each decoding operation would require a separate capture, loading and decoding sequence, as described above. Because of the time required to perform the capture and decoding steps as presently known, efficiency and practicality are each limited.

SUMMARY OF THE INVENTION

It is a primary object of the present invention to overcome the shortcomings of the prior art.

It is another primary object of the present invention to provide a process which utilizes real-time video obtained from a digital camera, endoscope, or other suitable image capturing device for continuous display and integrates the video capability in a computer to assess and decode any bar code readable information found in the video and display the encoded message, while simultaneously displaying the video image.

It is yet another primary object of the present invention to provide a physician, or other practitioner with an opportunity to alter the presentation of the video image or more preferably to aim the device in order to properly place a barcode readable symbol or symbols into the field of view of the imaging device and without having to separately upload

and download the scanning and decoding programs.

It is another primary object of the invention to allow a real-time captured video image to be stored in memory, such as for archival purposes.

It is another primary object of the present invention to allow a variety of various imaging devices of varying types, such as endoscopes, borescopes, and literally any digital camera capable of providing video input to a PC, to be interchangeably used with the above, irrespective of the types of imaging optics or illumination systems used therein.

It is another primary object of the present invention to provide a system which can automatically and without human intervention capture and decode a video input while allowing the real time image to remain displayed.

Therefore, and according to a preferred aspect of the present invention a process is provided for capturing and decoding bar-code information in real-time from a continuously displayed video signal, comprising the steps of:

aiming an imaging apparatus at a target of interest, said target having optically readable and machine readable information contained thereupon;

continually displaying a real time image of said target from said imaging apparatus on a computer video monitor;

selectively capturing an image into the memory of said computer;

decoding said image if said bar-code readable information is contained on said real-time image while maintaining said image on said display; and

outputting the decoded information.

According to a preferred embodiment, a captured image with or without bar-code data can be saved such as for archival purposes.

This application deals with integrating software which allows a personal computer (PC) or other peripheral device to be linked with literally any form of image capturing apparatus, such as an endoscope or other medical or other diagnostic instrument having a solid-state imaging device. A video image from the imaging device is displayed on the PC monitor while the system automatically captures the image, scans the image, and decodes and displays an encoded message, if such information is present.

The device is most useful for physicians or others who use endoscopic or other diagnostic instruments having video capability. The present process allows a real-time video image to be repeatedly and selectively captured to allow automatic decoding and outputting of encoded information. Furthermore, the user can conveniently aim, capture and decode in a single operation.

Other objects, features, and advantages will become apparent form the following Detailed Description of the Invention, when read in combination with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a preferred embodiment of the present invention;

FIG. 2 is a flow chart of the major process using a bar code reader and in accordance with a preferred aspect of the present invention;

FIG. 3 is a perspective view of one embodiment of a bar code reader which is used with the present invention;

FIG. 4 is a flow chart of one embodiment of the 2D portion of the auto discrimination process according to the invention;

FIG. 5 is a flow chart of one embodiment of the 1D portion of autodiscrimination program of the invention;

FIGS. 6 through 9 show representative bar code symbols of types which may be decoded by the present invention;

FIGS. 10 through 14 are drawings which aid in the understanding of the flow chart of FIG. 4; and

FIG. 15 is a copy of an embodiment of the source code useful for the process defined in the main program of FIG. 2.

DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG.1, there is shown a block diagram according to a preferred embodiment of the present invention. An imaging assembly 10 is provided for receiving an image of an object T and generating an electrical output signal indicative of the data optically encoded thereon, if any. The imaging assembly 10 may include an image sensor 14, such as a 1D or 2D CCD or CMOS solid state image sensor together with an imaging optics assembly 16 for receiving and focusing an image of the object T onto a substrate of the image sensor 14. Particular assemblies of this type are described in U.S. patent nos. 4,755,873, and 4,651,202, each of which are incorporated by reference. It will be readily apparent from the description of the invention, however, that literally any device having an internal image sensor is applicable. For purposes of this application, an electronic endoscope 60, such as described in the '202 reference is shown in Fig. 3.

The imaging device 14 also includes electronics which allow interfacing with a video capture card, also referred to as a frame grabber 18, which interfaces directly with a computer 20. According to this embodiment, the computer 20 is a Compaq Pentium 120 based PC and the frame grabber 18 is a Flashpoint Lite manufactured by Integral Technologies, Inc. These items are each well known in the art and require no further detailed discussion.

The computer includes a microprocessor 22 which is a programmable control device which is able to receive, output and process data in accordance with a stored program maintained within either or both of a read/write random access memory (RAM) 24 and a hard drive 26. The described embodiment preferably refers to a computer having the Microsoft Windows operating system contained therein, though other similar systems can be utilized.

The RAM 24 and the hard drive 26 are both connected to a common bus 28 through which program data, including address data, may be received and transmitted in either direction to any circuitry that is also connected thereto.

Included along the common bus 28 are user inputs, such as a keyboard/ mouse arrangement 34 as well as a video monitor 30 used to output the video signal, as is described in greater detail below.

The overall operation of the system illustrated in FIG. 1, will now be described with reference to the main program represented by the flow chart of FIG. 2.

Referring to FIG. 2, the main program begins with block 105 which includes the activation of the program instructions which can be accomplished by calling up the program from the hard drive 26 and loading the program into the system using the standard Windows commands from either the file manager or otherwise as is known. Upon loading of the program, a set of system parameters defining initial sizes of the frame window for the framegrabber 18, and other software variables, such as the mode of decoding, of triggering, etc are each activated. Each of the above parameters are described in greater detail in the attached source code, attached as FIG. 15 hereto.

In the initial start-up mode, a particular user input is selected as the trigger for the image capture and decode mechanism. According to this embodiment, a designated keyboard key, referred to as a "Hotkey", such as F10, is preset as the trigger. Alternately, the user can be prompted as described in greater detail according to the source code, FIG. 15, to utilize other keys on the keyboard or the mouse button as the trigger. Another alternate mode can be used in which a snapshot can be taken automatically by the processor 22 after a predetermined time interval has elapsed, e.g. 2 seconds. Similarly, the initial startup mode provides for outputting of any decoded messages to the monitor 30 after all of the symbols found have been decoded, such as adjacent the video image in a Windows message box. Alternate outputting modes, however, are contemplated allowing for several options for the user, such as saving to a ASCII file, sending the messages to a host processor, a keyboard buffer, or other convenient means.

The imaging assembly 10, using the imaging optics 16 focusses an image of the target T onto the image sensor 14 as is commonly known. The image of the target is then converted into analog electrical signals which are transmitted to the frame grabber 18 having circuitry to perform the analog to digital conversion to provide an array of pixels as defined by default parameters which is displayed as a continuous video signal in the monitor 30 per block 120. Details relating to the above are generally known to those of skill in the field, and require no further elaboration herein. The frame grabber 18 is preferably supplied as an input card to the computer 20 and serves as the video driver, though the frame grabber 18 can also be more than one driver, also as is commonly known.

Referring to the flow chart of FIG. 2, and after the video image of the target has been supplied by the image sensor to the monitor 30 per block 120, the user per block 125 is ready to capture an instantaneous image (hereinafter referred to as a "snapshot") for storage into RAM 24 per block 130 and for attempting to decode any symbols present in the field of view per block 135. As noted above, the initial mode contemplates the user must elect the capture of a snapshot using the Hotkey, F10, to execute block 125, though as also noted above, a number of alternate modes are available by which the user or the processor 22 may capture a snapshot. Regardless of the mode selected, the user aims the image sensor 16 at the object T until a resolved image is displayed in the monitor 30. Upon depression of the "hotkey", the processor 22 proceeds automatically to blocks 130 and 135, which call for the capture and attempt to decode an instantaneous image of the signal, referred to hereafter as a "snapshot", referring to an instantaneous digital signal which is stored into RAM. In passing, it should be noted that the acquisition of the snapshot of the video image does not impact the video image which remains displayed on the monitor, as a real time image.

Image capture of the snapshot to RAM 24 automatically causes the processor 22 to attempt to decode the snapshot as shown in the blocks 130, 135 in FIG. 2. This decoding involves a discrimination process, depending on whether any 1D and or 2D symbols are present in the field of view. This autodiscrimination process is described in greater detail below. If the decoding is successful, per the decision block 140, the messages are outputted per block 145. If the attempt to decode the messages is unsuccessful, the user may opt to recapture the video signal per block 125, and block 145 is bypassed. The user may alternately elect to store the captured image per block 150, into the hard disk 26.

Generally speaking, the flow chart of FIG. 4 illustrates the 1D portion of a complete 1D/2D autodiscrimination process, while FIG. 5 illustrates the 2D portion of the complete 1D/2D autodiscrimination process.

Turning first to the flow chart of FIG. 4, there is shown the 1D portion of the autodiscrimination process, which operates on the image data stored in RAM 24. The image data will comprise a gray scale representation of the 2D image formed on the image sensor.

On encountering block 305, the processor 22 is directed to calculate the "activities" of selected image data elements. The "activity" of a point P as used herein comprises a measure of the rate of change of the image data over a small two dimensional portion of the region surrounding point P. This activity is preferably calculated along any two arbitrarily selected directions which are mutually perpendicular to one another, as shown by the lines parallel to directions X and Y of FIG. 10. One example of an activity calculation is that which is based on the squares of the gray scale differences of two pairs of points P1X - P2X and P1Y - P2Y that are centered on point P, as shown in FIG. 10. Two mutually independent directions are used because the orientation of the symbol is unknown and because a high activity level that by chance is difficult to detect in a first direction will be readily detectable in a second direction perpendicular to the first direction.

Preferably, an activity profile of the stored image data is constructed on the basis of only a selected, relatively small number of image data elements that are distributed across the field of view that corresponds to the stored image data. Using a relatively small number of data elements is desirable to increase the speed at which the symbol can be imaged. These selected points may be selected as the points which lie at the intersections of an X-Y sampling grid such as that shown in FIG. 10. The spacing of the lines defining this grid is not critical to the present invention, but does affect the resolution with which the activity profile of the image can be measured.

When the processor 22 has determined the activities of the selected data points, it is directed to block 315, which causes it to look for candidate bar code symbols by identifying regions of high activity. This is conveniently done by determining those sets of image data points having activities which exceed a predetermined threshold value. A simplified, one-dimensional representation of this step is illustrated in FIG. 11, wherein those image data points having an activity that exceed a threshold value TH are labeled as a candidate symbol region CSR1.

In embodiments which are adapted to find and decode all of the symbols that occur in fields of view that include a plurality of bar code symbols, the result is the identification of a plurality of candidate symbol regions (CSRs), any one or more of which may be a bar code symbol. Whether or not they are bar code symbols is determined on the basis of whether they are decodable. According to this embodiment, the processor is instructed to select one of the CSRs according to a suitable selection rule, such as the largest CSR first, the CSR nearest the center of the field of view first, the CSR with the highest total activity first, etc., and then attempt to decode each of the symbols, to attempt to decode the first symbol and stop, depending on whether or not the symbol has been decoded.

Once all of the CSRs have been located, the processor 22 is directed to block 320 which calls for the processor to select the largest (or most centrally located) as yet unexamined CSR for further processing, and then to proceed to block 325. This latter block then causes the processor 22 to find the centroid or the center of gravity of that CSR, before proceeding to block 330. An example of such a centroid is labeled C in FIG. 12. Because the steps involved in finding a centroid or center of gravity are well known, they will not be described in any detail herein.

On encountering block 330, the processor 22 is directed to examine the selected CSR by defining exploratory scan lines therethrough, determining the activity profile of the CSR along those scan lines, and selecting the scan line having the highest total activity. In the case of a 1D bar code symbol, this will be the direction most nearly perpendicular to the direction most nearly perpendicular to the direction of the bars, i.e., the optimum reading direction for a 1D symbol.

On exiting block 330, the processor 22 encounters blocks 335 and 340. The first of these blocks 335 scans a scan line counter to zero; the second block 340 defines an initial working scan line through the centroid in the previously determined direction of highest activity. The result of this operation is the definition, in the image data space representation of the CSR, of a working scan line, such as SC=0 in FIG. 12.

Once the initial scan line has been defined, the processor 22 is directed to block 345 to calculate, by interpolation from the image data of the CSR, the values of the sampling points that the along this scan line. This means that, for each sampling point on the initial scan line, the processor 22 will calculate what brightness the sampling point would have if its brightness were calculated on the basis of the weighted brightness contributions of the four nearest brightness points of the CSR. These contributions are illustrated by the dotted lines which join the sample point SP of FIG. 13 to the four nearest image data points DPA-DPD. So long as these sampling points are more closely spaced than the image data points, this interpolation procedure will be performed on a subpixel basis, and this will produce a usably

accurate representation of the image data along the scan line. The result of the subpixel interpolation of the sampling points on a representative scan line of this type is shown in FIG. 14. Because the particulars of the subpixel interpolation process are known to those skilled in the art, this process will not be further described herein.

Once the above-described scan line data has been calculated, the processor 22 is directed to block 350, which calls for it to binarize the scan line data, i.e., convert it to a two-state representation of the data which can be processed as a candidate for 1D decoding. One such representation is commonly known as a timer count representation. One particularly advantageous procedure for accomplishing this binarization process is disclosed in U.S. Patent No. 5,586,960, which is hereby incorporated by reference.

On exiting block 350, the processor 22 will be in possession of a potentially decodable two-state 1D representation of the CSR. It then attempts to decode this representation, as called for by block 355. This attempted decoding will comprise the trial application to the representation of one 1D decoding program after another until the latter is either decoded or determined to be undecodable. Because decoding procedures of the latter type are known to those skilled in the art, they will not be discussed in any further detail.

As the 1D autodiscrimination process is completed, the processor 22 is directed to decision block 360 which causes it to continue along one of two different paths, depending on whether or not decoding was successful. If the decoding was not successful, the processor 22 will be caused to loop back to block 340, via blocks 365 and 370, where it will be caused to generate a new scan line SC=0, but that passes either above or below centroid C. This looping back step may be repeated many times, depending on the "spacing" of the new scan lines, until the entire CSR has been examined for decodable 1D data. If the entire CSR has been scanned and there has been no successful decode, the processor 22 is directed to exit the above-described loop via block 375. As used herein, the term "parallel" is broadly used to refer to scan lines or paths which are similarly distorted (e.g. curvilinear) as a result of foreshortening effects or as a result of being imaged from a non-planar surface. Since compensating for such distorting effects is known, as indicated, for example, by U.S. Patent No. 5,396,054, it will not be discussed in further detail.

Block 375 serves to direct the processor 22 back to block 320 to repeat the above-described selection, scanning and binarizing steps for the next unexamined CSR, if one is present. If another CSR is not present, or if the processor's program calls for an attempt to decode only one CSR, block 375 causes the processor 22 to exit the flow chart of FIG. 4 to begin an attempt to decode the then current set of image data as a 2D symbol, in accordance with the flow chart of FIG. 5. If other CSRs are present, block 375 directs the processor 22 back to block 320 to repeat the selection, scanning and binarizing process to the next CSR, and the next, and so on, until there is either a successful decode (block 360) or all of the CSRs have been examined (block 375).

If the processing of the first CSR has resulted in a successful decode, block 360 directs the processor 22 to block 380, which causes it to determine whether the decoded data indicates that the CSR contains a 1D stacked symbol, such as a PDF417 symbol. One example of such a symbol is shown in FIG. 9. If it is not, i.e., if the decoded symbol includes only a single row of bars, the 1D data is stored for later outputting in accordance with block 145 of the main program of FIG. 2, as called for by block 385. Alternately, the data may be output immediately and block 145 later skipped over. Then, if there are no remaining unexamined CSRs, the processor is directed to exit the flow chart of FIG. 4 via block 390. If, however, there are CSRs remaining, block 390 will direct the processor back to block 320 to begin processing the next CSR, and the next, and so on until all CSRs have been examined and decoded (block 390) or examined and found to be undecodable (block 375).

If, on encountering block 380, the decoded data indicates that the CSR contains a 1D stacked symbol, the above-described processing is modified by providing for the repetition of the scanning-digitizing process, beginning with block 340. This is accomplished by blocks 395, 396, and 397 in a manner that will be apparent to those skilled in the art. Significantly, by beginning the repeating of the process at block 340, all additional scan lines defined via the latter path will be parallel to the first decodable scan line, as required by a 1D stacked symbol, at least in the broad sense discussed earlier.

In view of the foregoing, it will be seen that, depending on the number of CSRs present, the flow chart will cause all 1D symbols in the image data to either be decoded or found to be undecodable before directing the processor 22 to exit the same.

The 2D autodiscrimination flow chart of FIG. 5 may be processed after the processing of the 1D autodiscrimination flow chart of FIG. 4 has been completed. It may also be processed without the flow chart of FIG. 4 having been processed, i.e., the 1D portion of the 1D/2D autodiscrimination process may be skipped or by passed. (In principle, the steps of the 2D portion of the 1D/2D discrimination process (FIG. 5) may also be processed before the 1D portion thereof (FIG. 4). Such arrangements are well within the intended scope of the present invention.

Referring to FIG.5, there is shown a flow chart of the 2D portion of the 1D/2D autodiscrimination process. When the flow chart of FIG. 5 is entered, the image data that is stored in RAM 24 is the same as that which would be stored therein if the flow chart of FIG. 4 were being entered. This data will comprise an array of 8-bit gray scale image data elements produced by the image sensor 16 and its associated signal and processing and A/D converter circuits contained in the framegrabber 18.

The flow chart of FIG.5 begins with a block 505, which directs the processor 22 to convert the gray image data representation stored in RAM 24 into a two-state or binarized representation of the same data. This may be accomplished in generally the same manner described earlier in connection with FIG. 11, i.e., by comparing these gray scale values to a threshold value and categorizing these values as 1s or 0s, depending on whether they exceed that threshold value.

Once the image data has been binarized, the processor 22 continues on to block 510, which causes it to identify and locate all of the 2D finder patterns that appear in the field of view of the image data. This is preferably accomplished by examining all of the candidate 2D finder patterns (CFPs) that are present and identifying them by type, i.e., identifying whether or not they are bulls eye type finder patterns, waistband type finder patterns or peripheral type finder patterns. An example of a bulls eye type finder pattern is shown in the central portion of the 2D bar code symbol of FIG. 6, which symbol encodes data in accordance with a 2D matrix symbology named "Aztec". An example of a waistband type finder pattern is shown in the middle portion of the 2D bar code symbol of FIG. 7, which symbol encodes data in accordance with a 2D matrix symbology named "Code One". An example of a peripheral type finder pattern is shown in the left and lower edges of the 2D bar code symbol of FIG. 8, which symbol encodes data in accordance with a 2D matrix symbology known as "Data Matrix". The finder identification processing is performed by applying to each CFP, in turn, a series of finder pattern finding algorithms of the type associated with each of the major types of finder patterns. Since such finder finding algorithms are known for finders of the waistband and peripheral types, these algorithms will not be discussed in further detail. One example of a finder finding algorithm for a waistband type finder may be found, for example, in "Uniform Symbology Specification Code One", published by AIM/USA Technology Group. Finder finding algorithms for bulls eye type finders that include concentric rings, (e.g. MaxiCode) are also known and therefore will also not be described in further detail.

Particularly advantageous for purposes of this embodiment, is a bulls eye finder finding algorithm of the type that may be used both with 2D symbologies, such as MaxiCode, that have bulls eye finder patterns that include concentric rings and with 2D symbologies, such as Aztec, that have bulls eye finder patterns which include concentric polygons. A finder finding algorithm of the latter type is described in described in copending and commonly assigned USSN 08/504,643, which has been incorporated herein by reference. The Aztec 2D bar code symbology itself is fully described in USSN 08/441,446, which has also been incorporated by reference.

Once all of the finder patterns have been located and their types have been determined, the processor 22 is directed to decision block 515. This block affords the processor an opportunity to exit the flop chart of FIG. 5, via exit block 545, if no 2D finder patterns could be found and typed. This block speeds up the execution of the program by skipping over decoding operations which have no hope of success without their associated finder pattern.

If a finder pattern has been found and typed, the processor 22 is directed to block 525. This block causes the processor 22 to select for decoding the bar code symbol whose finder is closest to the center of field of view of the image data. Optionally, the processor may be instructed to find the largest 2D bar code symbol that uses a particular 2D symbology or the 2D bar code symbol using a particular 2D symbology which is closest to the center of the field of view of the image data. Once this selection is made, the processor attempts to decode that symbol, as called for block 530. If this decoding attempt is successful, as determined by decision block 535, the resulting data may be stored for outputting in accordance with block 135 of the main program of FIG. 2, as called for by block 540. Alternately, the decoded data may be outputted immediately and block 135 later skipped over. If the decoding attempt is not successful, however, block 540 is skipped, and the processor 22 is directed to decision block 545.

Block 545 will direct the processor 22 back to block 525 to process the next 2D symbol, i.e., the symbol whose CFR is next closest to the center of the field of view. The above-described attempted decoding and storing (or outputting) steps will then be repeated, one CFR after another, until there are no more symbols which have usable finder data patterns. Finally, when all symbols having usable finder patterns have either been decoded or found to be undecodable, the processor will exit the flow chart of FIG. 5, via block 550, to return to the main program of FIG. 2.

All of the decoded messages are then outputted to the monitor 30, and the entire output is displayed. According to this embodiment, the contents of each of the bar code messages are output to the WINDOWS message box, though other alternatives are possible.

The invention has been described with reference to a specific embodiment, but it will be readily apparent that other similar embodiments can be utilized to provide decoding and outputting of a real time video signal and as covered by the accompanying claims.

ANNEX TO THE DESCRIPTION

C:\VIDDCODE\MAINFRM.CPP

```
/*****************************************************************************
 *
 *   captevw.cpp: implementation of the CMainFrame class
 *
 *   Microsoft Video for Windows Capture Class Sample Program
 *
 *****************************************************************************/
/*****************************************************************************
 *
 *  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 *  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
 *  PURPOSE.
 *
 *  Copyright (c) 1992, 1993  Microsoft Corporation.  All Rights Reserved.
 *
 *****************************************************************************/

#include "stdafx.h"
#include "videcode.h"
#include "mainfrm.h"
#include "viddoc.h"
#include "vidvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_COMMAND(ID_WINDOW_TILE_HORZ, OnWindowTileHorz)
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYENDSESSION()
   ON_WM_CHAR()
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_FILE_NEW,
        ID_FILE_OPEN,
        ID_FILE_SAVE,
                ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
                ID_SEPARATOR,
        ID_FILE_PRINT,
        ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
```

C:\VIDDCODE\MAINFRM.CPP

```
{
        ID_SEPARATOR,                       // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

   if (!m_wndToolBar.Create(this) ||
      !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
      !m_wndToolBar.SetButtons(buttons,
      sizeof(buttons)/sizeof(UINT)))
   {
      TRACE("Failed to create toolbar\n");
      return -1;                  // fail to create
   }

   if (!m_wndStatusBar.Create(this) ||
       !m_wndStatusBar.SetIndicators(indicators,
       sizeof(indicators)/sizeof(UINT)))
   {
      TRACE("Failed to create status bar\n");
      return -1;                  // fail to create
   }
   return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

//=============================================================
// Append a list of current capture drivers to our menu
//=============================================================
void CMainFrame::UpdateDriverMenu(CCapWnd &rwndCap)
{
```

C:\VIDDCODE\MAINFRM.CPP

```
    int j;
    CMenu*    pMenu;
    CMenu*    pSubMenu;

    pMenu = GetMenu();
    ASSERT(pMenu != NULL);

    pSubMenu = pMenu->GetSubMenu(3);
    ASSERT(pSubMenu != NULL);

    TRACE("Updating Driver Menu");

    for (j =0; j < 10; j++)
    {
       // Initially, all 10 selections are in the menu
       // to enable the auto status bar messages
       pSubMenu->DeleteMenu(ID_CAP_DRV0 + j, MF_BYCOMMAND);
       if(rwndCap.DriverConnect(j))
       {
          pSubMenu->AppendMenu(MF_ENABLED, ID_CAP_DRV0 + j, rwndCap.DriverGetName()) ;
          rwndCap.DriverDisconnect();
       }
    }
}

////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnWindowTileHorz()
{
   MDITile (MDITILE_VERTICAL);
}

void CMainFrame::StatusCallback (int nID, LPCSTR lpszStatusText)
{
   m_wndStatusBar.SetPaneText( 0 /* nIndex */,  lpszStatusText, TRUE );
   m_wndStatusBar.UpdateWindow();
}

void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    SendMessageToDescendants( WM_PALETTECHANGED, 0, 0, TRUE);
}

BOOL CMainFrame::OnQueryEndSession()
{
    if (!CMDIFrameWnd::OnQueryEndSession())
        return FALSE;

    CMDIChildWnd* pMDIChildWnd = MDIGetActive();
    if (pMDIChildWnd == NULL)
        return TRUE; // no active MDI child frame

    CView* pView = pMDIChildWnd->GetActiveView();
    ASSERT(pView != NULL);

    // Ask the active MDI child if its OK to exit...
    return (BOOL) pView->SendMessage(WM_QUERYENDSESSION, 0, 0);
}

void CMainFrame::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
   TRACE("CMainFrame Onchar nChar=%d, repeat=%d, flags=%d\n",nChar,nRepCnt,nFlags);
```

C:\VIDDCODE\MAINFRM.CPP

```
    CMDIFrameWnd::OnChar(nChar, nRepCnt, nFlags);
}
```

Page: 4

C:\VIDDCODE\MAINFRM.H

```
// mainfrm.h : interface of the CMainFrame class
//
//
//  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
//  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
//  PURPOSE.
//
//  Copyright (c) 1992, 1993  Microsoft Corporation.  All Rights Reserved.
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CMDIFrameWnd
{
        DECLARE_DYNAMIC(CMainFrame)
public:
   CMainFrame();
   void            StatusCallback (int nID, LPCSTR lpStatusText);

// Attributes
public:

// Operations
public:
   void            UpdateDriverMenu(CCapWnd& rwndCap);

// Implementation
public:
   virtual         ~CMainFrame();
#ifdef _DEBUG
   virtual void    AssertValid() const;
   virtual void    Dump(CDumpContext& dc) const;
#endif


protected:
   CStatusBar      m_wndStatusBar;
   CToolBar        m_wndToolBar;

// Generated message map functions
protected:
   //{{AFX_MSG(CMainFrame)
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
   afx_msg void OnWindowTileHorz();
   afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
   afx_msg BOOL OnQueryEndSession();
   afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

C:\VIDDCODE\MSGDLG.CPP

```
// msgdlg.cpp : implementation file
//

#include "stdafx.h"
#include "msgdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMsgDlg dialog


CMsgDlg::CMsgDlg(CString strMsg /*=NULL*/, CWnd* pParent /*=NULL*/)
   : CDialog(CMsgDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CMsgDlg)
   m_strMsg = strMsg;
   //}}AFX_DATA_INIT
}

void CMsgDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CMsgDlg)
   DDX_Text(pDX, IDC_EDIT1, m_strMsg);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMsgDlg, CDialog)
   //{{AFX_MSG_MAP(CMsgDlg)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CMsgDlg message handlers

BOOL CMsgDlg::OnInitDialog()
{
   CDialog::OnInitDialog();
   CenterWindow();
   return TRUE;  // return TRUE  unless you set the focus to a control
}
```

C:\VIDDCODE\MSGDLG.H

```
// msgdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CMsgDlg dialog

class CMsgDlg : public CDialog
{
// Construction
public:
   CMsgDlg(CString strMsg = NULL, CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CMsgDlg)
   enum { IDD = IDD_DLG_MSG };
   CString  m_strMsg;
   //}}AFX_DATA

// Implementation
protected:
   virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support

   // Generated message map functions
   //{{AFX_MSG(CMsgDlg)
   virtual BOOL OnInitDialog();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};
```

C:\VIDDCODE\PREFSDLG.CPP

```
// prefsdlg.cpp : implementation file
//

#include "stdafx.h"
#include "videcode.h"
#include "prefsdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPrefsDlg dialog


CPrefsDlg::CPrefsDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CPrefsDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CPrefsDlg)
   m_strOutFile = "";
   m_bDecodeHotKey = FALSE;
   m_bDecodeMouse = FALSE;
   m_bDecodeTimer = FALSE;
   m_DecodeInterval = 0;
   m_OutputSel = 0;
   //}}AFX_DATA_INIT
}

void CPrefsDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CPrefsDlg)
   DDX_Text(pDX, IDC_FILE, m_strOutFile);
   DDX_Check(pDX, IDC_DEC_KB, m_bDecodeHotKey);
   DDX_Check(pDX, IDC_DEC_MOUSE, m_bDecodeMouse);
   DDX_Check(pDX, IDC_DEC_TIMER, m_bDecodeTimer);
   DDX_Text(pDX, IDC_DEC_INTERVAL, m_DecodeInterval);
   DDV_MinMaxUInt(pDX, m_DecodeInterval, 0, 60000);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPrefsDlg, CDialog)
   //{{AFX_MSG_MAP(CPrefsDlg)
   ON_BN_CLICKED(IDC_OUT_FILE, OnOutFile)
   ON_BN_CLICKED(IDC_OUT_CLIP, OnOutClip)
   ON_BN_CLICKED(IDC_OUT_EDIT, OnOutEdit)
   ON_BN_CLICKED(IDC_OUT_MSG, OnOutMsg)
   ON_BN_CLICKED(IDC_DEFAULT, OnDefault)
   ON_BN_CLICKED(IDC_DEC_TIMER, OnDecodeTimer)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CPrefsDlg message handlers

BOOL CPrefsDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // Initialize radio buttons
   m_strOutFile = MfxGetApp()->GetProfileString("Settings","Output File",DEFAULT_OUT_FILE);
   m_OutputSel = MfxGetApp()->GetProfileInt("Settings","Output",DEFAULT_OUT_MODE);
```

C:\VIDDCODE\PREFSDLG.CPP

```
    m_bDecodeHotKey = MfxGetApp()->GetProfileInt("Settings","Decode Hot Key",DEFAULT_DEC_KB);
    m_bDecodeMouse = MfxGetApp()->GetProfileInt("Settings","Decode Mouse",DEFAULT_DEC_MOUSE);
    m_bDecodeTimer = MfxGetApp()->GetProfileInt("Settings","Decode Timer",DEFAULT_DEC_TIMER);
    m_DecodeInterval = MfxGetApp()->GetProfileInt("Settings","Decode Interval",DEFAULT_DEC_INTERVAL

    Display();
    CenterWindow();

    return TRUE;  // return TRUE  unless you set the focus to a control
}

//=============================================
// Setup all variables and profile settings
// based on the current selections.
//=============================================
void CPrefsDlg::OnOK()
{
    // Transfer all dialog data to class members.
    UpdateData();

    //---------------------------------------
    // Determine which radio button is checked
    //---------------------------------------
    if(((CButton*)GetDlgItem(IDC_OUT_MSG))->GetCheck())
    {
        MfxGetApp()->WriteProfileInt("Settings","Output",IDC_OUT_MSG);
    }
    else if(((CButton*)GetDlgItem(IDC_OUT_CLIP))->GetCheck())
    {
        MfxGetApp()->WriteProfileInt("Settings","Output",IDC_OUT_CLIP);
    }
    else if(((CButton*)GetDlgItem(IDC_OUT_EDIT))->GetCheck())
    {
        MfxGetApp()->WriteProfileInt("Settings","Output",IDC_OUT_EDIT);
    }
    else if(((CButton*)GetDlgItem(IDC_OUT_FILE))->GetCheck())
    {
        MfxGetApp()->WriteProfileInt("Settings","Output",IDC_OUT_FILE);
    }

    // Write the output file to the ini file
    MfxGetApp()->WriteProfileString("Settings","Output File",m_strOutFile);

    //-------------------------------------------------
    // Write the decode mode
    //-------------------------------------------------
    MfxGetApp()->WriteProfileInt("Settings","Decode Hot Key",m_bDecodeHotKey);
    MfxGetApp()->WriteProfileInt("Settings","Decode Mouse",m_bDecodeMouse);
    MfxGetApp()->WriteProfileInt("Settings","Decode Timer",m_bDecodeTimer);
    MfxGetApp()->WriteProfileInt("Settings","Decode Interval",m_DecodeInterval);

    CDialog::OnOK();
}

//=====================================================
// This group of methods used to set proper state of
// the filename edit text box.
//=====================================================
void CPrefsDlg::OnOutFile()
{
    CEdit*   pEditFileName = (CEdit*)GetDlgItem(IDC_FILE);
    pEditFileName->EnableWindow(TRUE);
    m_OutputSel = IDC_OUT_FILE;
}
```

C:\VIDDCODE\PREFSDLG.CPP

```
void CPrefsDlg::OnOutClip()
{
   CEdit*   pEditFileName = (CEdit*)GetDlgItem(IDC_FILE);
   pEditFileName->EnableWindow(FALSE);
   m_OutputSel = IDC_OUT_CLIP;
}
void CPrefsDlg::OnOutEdit()
{
   CEdit*   pEditFileName = (CEdit*)GetDlgItem(IDC_FILE);
   pEditFileName->EnableWindow(FALSE);
   m_OutputSel = IDC_OUT_EDIT;
}
void CPrefsDlg::OnOutMsg()
{
   CEdit*   pEditFileName = (CEdit*)GetDlgItem(IDC_FILE);
   pEditFileName->EnableWindow(FALSE);
   m_OutputSel = IDC_OUT_MSG;
}

//---------------------------------------------------------
// Setup the default states of all controls
//---------------------------------------------------------
void CPrefsDlg::OnDefault()
{
   CButton*    pButton = (CButton*)GetDlgItem(m_OutputSel);
   pButton->SetCheck(FALSE);

   m_strOutFile = DEFAULT_OUT_FILE;
   m_bDecodeHotKey = DEFAULT_DEC_KB;
   m_bDecodeMouse = DEFAULT_DEC_MOUSE;
   m_bDecodeTimer = DEFAULT_DEC_TIMER;
   m_DecodeInterval = DEFAULT_DEC_INTERVAL;
   m_OutputSel = DEFAULT_OUT_MODE;

   Display();

}

//---------------------------------------------------------
// Displays controls on dialog box according to current
// state of the variables.
//---------------------------------------------------------
void  CPrefsDlg::Display()
{
   // Display radio buttons
   CButton*    pButton = (CButton*)GetDlgItem(m_OutputSel);
   pButton->SetCheck(TRUE);

   // Display Output File Edit Window Control
   CEdit* pEdit = (CEdit*)GetDlgItem(IDC_FILE);
   pEdit->SetWindowText(m_strOutFile);
   if(m_OutputSel == IDC_OUT_FILE)
      pEdit->EnableWindow(TRUE);
   else
      pEdit->EnableWindow(FALSE);

   // Set Display States of the Decode mode options
   pButton = (CButton*)GetDlgItem(IDC_DEC_KB);
   pButton->SetCheck(m_bDecodeHotKey);
   pButton = (CButton*)GetDlgItem(IDC_DEC_MOUSE);
   pButton->SetCheck(m_bDecodeMouse);
   pButton = (CButton*)GetDlgItem(IDC_DEC_TIMER);
   pButton->SetCheck(m_bDecodeTimer);
```

C:\VIDDCODE\PREFSDLG.CPP

```
   // Set Decode Timer Interval Edit box
   pEdit = (CEdit*)GetDlgItem(IDC_DEC_INTERVAL);
   char  Buff[10];
   itoa(m_DecodeInterval,Buff,10);
   pEdit->SetWindowText(Buff);
   pEdit->EnableWindow(m_bDecodeTimer);
}

//---------------------------------------------------------
// When DecodeTimer mode is enabled then enable the
// Time interval edit control.
//---------------------------------------------------------
void CPrefsDlg::OnDecodeTimer()
{
   CEdit*   pEdit = (CEdit*)GetDlgItem(IDC_DEC_INTERVAL);
   CButton* pButton = (CButton*)GetDlgItem(IDC_DEC_TIMER);
   pEdit->EnableWindow(pButton->GetCheck());
}
```

C:\VIDDCODE\PREFSDLG.H

```
// prefsdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPrefsDlg dialog

class CPrefsDlg : public CDialog
{
// Construction
public:
   CPrefsDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CPrefsDlg)
   enum { IDD = IDD_PREFS };
   CString  m_strOutFile;
   BOOL  m_bDecodeHotKey;
   BOOL  m_bDecodeMouse;
   BOOL  m_bDecodeTimer;
   UINT  m_DecodeInterval;
   UINT  m_OutputSel;
   //}}AFX_DATA

// Implementation
protected:
   void         Display();
   virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support

   // Generated message map functions
   //{{AFX_MSG(CPrefsDlg)
   virtual BOOL OnInitDialog();
   virtual void OnOK();
   afx_msg void OnOutFile();
   afx_msg void OnOutClip();
   afx_msg void OnOutMsg();
   afx_msg void OnOutEdit();
   afx_msg void OnDefault();
   afx_msg void OnDecodeTimer();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};
```

Page: 1

C:\VIDDCODE\RESOURCE.H

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by VIDECODE.RC
//
#define IDR_MAINFRAME                   2
#define IDR_CAPTESTYPE                  3
#define IDR_VIDECODE_DOC                3
#define IDD_ABOUTBOX                    100
#define IDD_SETUP_DIALOG                101
#define IDC_AUDIO                       102
#define IDD_DLG_MSG                     103
#define IDC_DOS_BUFFERS                 104
#define IDD_PREFS                       106
#define IDC_FRAME_RATE                  107
#define IDC_NUM_VIDEO_BUFFERS           108
#define IDC_VIDEO_BUFFERS               109
#define IDC_SETAUDIOFORMAT              112
#define IDC_AUDIOFORMATTEXT             113
#define IDC_SMARTDRV                    114
#define IDC_FILE                        117
#define IDC_OUT_EDIT                    118
#define IDC_OUT_MSG                     120
#define IDC_OUT_FILE                    121
#define IDC_OUT_CLIP                    122
#define IDC_DEC_MOUSE                   126
#define IDC_DEC_TIMER                   127
#define IDC_DEC_KB                      128
#define IDC_DEC_INTERVAL                129
#define IDC_DEFAULT                     130
#define ID_CAP_DRV0                     200
#define ID_CAP_DRV1                     201
#define ID_CAP_DRV2                     202
#define ID_CAP_DRV3                     203
#define ID_CAP_DRV4                     204
#define ID_CAP_DRV5                     205
#define ID_CAP_DRV6                     206
#define ID_CAP_DRV7                     207
#define ID_CAP_DRV8                     208
#define ID_CAP_DRV9                     209
#define IDC_EDIT1                       1001
#define ID_CAP_OVERLAY                  32770
#define ID_CAP_PREVIEW                  32771
#define ID_CAP_DLG_FORMAT               32772
#define ID_CAP_DLG_SOURCE               32773
#define ID_CAP_DLG_DISPLAY              32774
#define ID_CAP_AUTOPALS                 32776
#define ID_CAP_SEQUENCE                 32778
#define ID_CAP_SCALE                    32793
#define ID_CAP_AUDIO                    32794
#define ID_CAP_SETUP                    32795
#define ID_FILE_NEWPLAYBACKWINDOW       32796
#define ID_CAP_PLAYBACK                 32798
#define ID_FILE_PREFERENCES             32799
#define ID_ACCEL_DECODE                 32801

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        107
#define _APS_NEXT_COMMAND_VALUE         32802
#define _APS_NEXT_CONTROL_VALUE         131
#define _APS_NEXT_SYMED_VALUE           112
```

Page: 1

C:\VIDDCODE\RESOURCE.H

```
#endif
#endif
```

C:\VIDDCODE\STDAFX.CPP

```
// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

Page: 1

C:\VIDDCODE\STDAFX.H

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//    are changed infrequently
//
//

#include <mfx.h>
#include <images.h>
#include <mfxvid.h>
#include <wadecode.h>
#include "resource.h"


#define  DEFAULT_OUT_FILE      "out.txt"
#define  DEFAULT_OUT_MODE      IDC_OUT_MSG
#define  DEFAULT_CAP_BMP       "cap.bmp"
#define  DEFAULT_DEC_MOUSE     TRUE
#define  DEFAULT_DEC_KB        FALSE
#define  DEFAULT_DEC_TIMER     FALSE
#define  DEFAULT_DEC_INTERVAL  5000
#define  TIMER1                1
#define  MAX_MSG_LEN           4096
```

Page: 1

C:\VIDDCODE\VIDDOC.CPP

```
/*****************************************************************************
 *
 *   VidDoc.cpp: implementation of the CVidecodeDoc class
 *
 *   Microsoft Video for Windows Capture Class Sample Program
 *
 *****************************************************************************/
/*****************************************************************************
 *
 *  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 *  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
 *  PURPOSE.
 *
 *  Copyright (c) 1992, 1993  Microsoft Corporation.  All Rights Reserved.
 *
 *****************************************************************************/

#include "stdafx.h"
#include "videcode.h"
#include "viddoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////
// CVidecodeDoc

IMPLEMENT_DYNCREATE(CVidecodeDoc, CDocument)

BEGIN_MESSAGE_MAP(CVidecodeDoc, CDocument)
        //{{AFX_MSG_MAP(CVidecodeDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////
// CVidecodeDoc construction/destruction

CVidecodeDoc::CVidecodeDoc()
{

}

CVidecodeDoc::~CVidecodeDoc()
{
}

BOOL CVidecodeDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        return TRUE;
}

////////////////////////////////////////////////////////////////////////////////
// CVidecodeDoc serialization

void CVidecodeDoc::Serialize(CArchive& ar)
```

C:\VIDDCODE\VIDDOC.CPP

```
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CVidecodeDoc diagnostics

#ifdef _DEBUG
void CVidecodeDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CVidecodeDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CVidecodeDoc commands
```

C:\VIDDCODE\VIDDOC.H

```
// VidDoc.h : interface of the CVidecodeDoc class
//
/////////////////////////////////////////////////////////////////////////////
#ifndef _VIDDOC_H_
#define _VIDDOC_H_

class CVidecodeDoc : public CDocument
{
protected: // create from serialization only
   CVidecodeDoc();
   DECLARE_DYNCREATE(CVidecodeDoc)

// Attributes
public:

// Operations
public:

// Implementation
public:
   virtual -CVidecodeDoc();
   virtual void Serialize(CArchive& ar);  // overridden for document i/o
#ifdef _DEBUG
   virtual  void AssertValid() const;
   virtual  void Dump(CDumpContext& dc) const;
#endif
protected:
   virtual  BOOL  OnNewDocument();

// Generated message map functions
protected:
   //{{AFX_MSG(CVidecodeDoc)
      // NOTE - the ClassWizard will add and remove member functions here.
      //    DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

#endif   // _VIDDOC_H_
```

C:\VIDDCODE\VIDECODE.CPP

```
/*
 *   Videcode.cpp: Defines the class behaviors for the application.
 *
 */

#include "stdafx.h"
#include "videcode.h"
#include "mainfrm.h"
#include "viddoc.h"
#include "vidvw.h"
#include "prefsdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CVidecodeApp

BEGIN_MESSAGE_MAP(CVidecodeApp, CMfxApp)
    //{{AFX_MSG_MAP(CVidecodeApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
   ON_COMMAND(ID_FILE_PREFERENCES, OnFilePreferences)
   //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CMfxApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CMfxApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVidecodeApp construction

CVidecodeApp::CVidecodeApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CVidecodeApp object

CVidecodeApp NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
// CVidecodeApp initialization

BOOL CVidecodeApp::InitInstance()
{
   // Initialize MFX app (Ctl3dv2)
   CMfxApp::InitInstance();

   // Standard initialization
   LoadStdProfileSettings();  // Load standard INI file options (including MRU)

   // Register the application's document templates.  Document templates
   //  serve as the connection between documents, frame windows and views.
   AddDocTemplate(new CMultiDocTemplate(IDR_CAPTESTYPE,
            RUNTIME_CLASS(CVidecodeDoc),
            RUNTIME_CLASS(CMDIChildWnd),        // standard MDI child frame
            RUNTIME_CLASS(CVidecodeView)));

   // create main MDI Frame window
   CMainFrame* pMainFrame = new CMainFrame;
```

C:\VIDDCODE\VIDECODE.CPP

```
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    m_pMainWnd = pMainFrame;

    // Show an initial Capture Window
    OnFileNew();

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CVidecodeApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

void CVidecodeApp::OnFilePreferences()
{
   CPrefsDlg   dlg;
   dlg.DoModal();
```

C:\VIDDCODE\VIDECODE.CPP

```
}
```

C:\VIDDCODE\VIDECODE.H

```
// videcode.h : main header file for the videcode application
//
//

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CVideoApp:
//

class CVidecodeApp : public CMfxApp
{
public:
    CVidecodeApp();

// Overrides
    virtual BOOL InitInstance();

// Implementation

    //{{AFX_MSG(CVidecodeApp)
    afx_msg void OnAppAbout();
   afx_msg void OnFilePreferences();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

C:\VIDDCODE\VIDVW.CPP

```
/*****************************************************************************
 *
 *   Vidvw.cpp: implementation of the CVidecodeView class
 *
 *   Warning:   Some playback hardware does not support an active capture
 *              window simultaneous with an active playback window.
 *              For more information, see the comments in the
 *              CVidecodeView::OnCapPlayback method below.
 *
 *****************************************************************************/

#include "stdafx.h"
#include "videcode.h"
#include "viddoc.h"
#include "vidvw.h"
#include "mainfrm.h"
#include "cpprmdlg.h"
#include "msgdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////
// CVidecodeView
/////////////////////////////////////////////////
IMPLEMENT_DYNCREATE(CVidecodeView, CView)

BEGIN_MESSAGE_MAP(CVidecodeView, CView)
    //{{AFX_MSG_MAP(CVidecodeView)
    ON_WM_CREATE()
    ON_WM_SIZE()
    ON_COMMAND(ID_CAP_OVERLAY, OnCapOverlay)
    ON_COMMAND(ID_CAP_PREVIEW, OnCapPreview)
    ON_COMMAND(ID_CAP_DLG_FORMAT, OnCapDlgFormat)
    ON_COMMAND(ID_CAP_DLG_DISPLAY, OnCapDlgDisplay)
    ON_COMMAND(ID_CAP_DLG_SOURCE, OnCapDlgSource)
    ON_COMMAND(ID_CAP_AUTOPAL5, OnCapAutopal5)
    ON_COMMAND(ID_CAP_SEQUENCE, OnCapSequence)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_CAP_PREVIEW, OnUpdateCapPreview)
    ON_UPDATE_COMMAND_UI(ID_CAP_OVERLAY, OnUpdateCapOverlay)
    ON_COMMAND(ID_CAP_SCALE, OnCapScale)
    ON_UPDATE_COMMAND_UI(ID_CAP_SCALE, OnUpdateCapScale)
    ON_UPDATE_COMMAND_UI(ID_CAP_DLG_DISPLAY, OnUpdateCapDlgDisplay)
    ON_UPDATE_COMMAND_UI(ID_CAP_DLG_FORMAT, OnUpdateCapDlgFormat)
    ON_UPDATE_COMMAND_UI(ID_CAP_DLG_SOURCE, OnUpdateCapDlgSource)
    ON_COMMAND(ID_CAP_SETUP, OnCapSetup)
    ON_COMMAND(ID_CAP_DRV0, OnCapDrv0)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV0, OnUpdateCapDrv)
    ON_COMMAND(ID_CAP_DRV1, OnCapDrv1)
    ON_COMMAND(ID_CAP_DRV2, OnCapDrv2)
    ON_COMMAND(ID_CAP_DRV3, OnCapDrv3)
    ON_COMMAND(ID_CAP_DRV4, OnCapDrv4)
    ON_COMMAND(ID_CAP_DRV5, OnCapDrv5)
    ON_COMMAND(ID_CAP_DRV6, OnCapDrv6)
    ON_COMMAND(ID_CAP_DRV7, OnCapDrv7)
    ON_COMMAND(ID_CAP_DRV8, OnCapDrv8)
    ON_COMMAND(ID_CAP_DRV9, OnCapDrv9)
    ON_WM_SETFOCUS()
```

C:\VIDDCODE\VIDVW.CPP

```
    ON_WM_KILLFOCUS()
    ON_UPDATE_COMMAND_UI(ID_CAP_AUTOPAL5, OnUpdateCapAutopal5)
    ON_UPDATE_COMMAND_UI(ID_CAP_SEQUENCE, OnUpdateCapSequence)
    ON_UPDATE_COMMAND_UI(ID_CAP_SETUP, OnUpdateCapSetup)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_WM_QUERYENDSESSION ()
    ON_WM_LBUTTONDOWN()
   ON_WM_TIMER()
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV1, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV2, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV3, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV4, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV5, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV6, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV7, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV8, OnUpdateCapDrv)
    ON_UPDATE_COMMAND_UI(ID_CAP_DRV9, OnUpdateCapDrv)
   ON_COMMAND(ID_ACCEL_DECODE, OnAccelDecode)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////
// Declare the static members
////////////////////////////////
BOOL  CVidecodeView::s_bMenuSet = FALSE;


/////////////////////////////////////////////////
// CVidecodeView construction/destruction
/////////////////////////////////////////////////
CVidecodeView::CVidecodeView()
{
    m_fDialogIsUp = FALSE;
    m_TimerID = 0;
    m_strOldMsg = "";
}

CVidecodeView::~CVidecodeView()
{
   if(m_TimerID != 0)
      KillTimer(m_TimerID);
}

/////////////////////////////////////////////////
// CVidecodeView diagnostics

#ifdef _DEBUG
void CVidecodeView::AssertValid() const
{
    CView::AssertValid();
}

void CVidecodeView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CVidecodeDoc* CVidecodeView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CVidecodeDoc)));
    return (CVidecodeDoc*) m_pDocument;
}
```

C:\VIDDCODE\VIDVW.CPP

```
#endif //_DEBUG


// Append a list of current capture drivers to our menu
void CVidecodeView::GetDriverList()
{
   CString  Name;

   CMenu* pMenu = MfxGetApp()->m_pMainWnd->GetMenu();
   ASSERT(pMenu != NULL);
   CMenu* pSubMenu = pMenu->GetSubMenu(3);
   ASSERT(pSubMenu != NULL);

   // Initially, all 10 selections are in the menu
   // to enable the auto status bar messages
   for (int j =0; j < 10; j++)
   {
      pSubMenu->DeleteMenu(ID_CAP_DRV0 + j, MF_BYCOMMAND);
      if( m_wndCap.DriverConnect(j) )
      {
         Name =  m_wndCap.DriverGetName();
         if (Name.GetLength() > 0)
            pSubMenu->AppendMenu(MF_ENABLED, ID_CAP_DRV0 + j, Name);
         m_wndCap.DriverDisconnect();
      }
   }
}

////////////////////////////////////////////////
// Setup the capture interval timer
////////////////////////////////////////////////
void CVidecodeView::SetupTimer(UINT Interval)
{
    // Kill any existing Timers.
   if (m_TimerID != 0)
   {
      KillTimer(m_TimerID);
      m_TimerID = 0;
   }

   if( !(m_TimerID = SetTimer(TIMER1, Interval, NULL)) )
   {
      AfxMessageBox("Error:  No Timers Available", MB_ICONEXCLAMATION);
   }
}

////////////////////////////////////////////////
// Create a capture window
////////////////////////////////////////////////
int CVidecodeView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   static int nID = 1;
   int   nIndex;
   CRect rc;

   // Create the view window
   if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

   // Setup the timer if we're configured for interval decodes
   if( MfxGetApp()->GetProfileInt("Settings","Decode Timer",DEFAULT_DEC_TIMER) )
      SetupTimer( MfxGetApp()->GetProfileInt("Settings","Decode Interval",DEFAULT_DEC_INTERVAL) );

   // Create a new capture window and size it.
```

C:\VIDDCODE\VIDVW.CPP

```
   if(!m_wndCap.Create("", WS_CHILD | WS_VISIBLE, nID++, rc, this))
      return -1;

   // Execute this once and only once for the
   // entire view class on the first view creation
   if (!s_bMenuSet)
   {
      ResizeNow();
      GetDriverList();
      s_bMenuSet = TRUE;
   }

   // Connect to the next available driver
   for (nIndex = 0; nIndex < 10; nIndex++)
      if (m_wndCap.DriverConnect(nIndex))
         break;
   m_nDriverIndex = ((nIndex == 10) ? -1 : nIndex);

   // Find out what the driver can do
   if (!m_wndCap.DriverGetCaps(m_CapDriverCaps))
   {
      _fmemset (&m_CapDriverCaps, 0, sizeof (m_CapDriverCaps));
      _fmemset (&m_CapStatus, 0, sizeof (m_CapStatus));
   }

   // Retrieve the default capture settings
   m_wndCap.CaptureGetSetup(m_CaptureParms);
   ResizeNow();

   m_wndCap.PreviewRate(66);        // Set the preview rate to 66 milliseconds
   if (m_CapDriverCaps.fHasOverlay)    // Enable overlay if the card supports it
   {
      m_wndCap.Overlay(TRUE);
      m_wndCap.Preview(FALSE);
   }
   else
   {
      m_wndCap.Preview(TRUE);
      m_wndCap.Overlay(FALSE);
   }

   return 0;
}

//////////////////////////////////////////////
// Preview and Overlay selection
//////////////////////////////////////////////
void CVidecodeView::OnCapOverlay()
{
     m_wndCap.GetStatus(m_CapStatus);
     m_CapStatus.fOverlayWindow = !m_CapStatus.fOverlayWindow;
     m_CapStatus.fLiveWindow = !m_CapStatus.fOverlayWindow;
     m_wndCap.Overlay(m_CapStatus.fOverlayWindow);
     m_wndCap.Preview(m_CapStatus.fLiveWindow);
}

void CVidecodeView::OnCapPreview()
{
     m_wndCap.GetStatus(m_CapStatus);
     m_CapStatus.fOverlayWindow = !m_CapStatus.fOverlayWindow;
     m_CapStatus.fLiveWindow = !m_CapStatus.fLiveWindow;
     m_wndCap.Overlay(m_CapStatus.fOverlayWindow);
     m_wndCap.Preview(m_CapStatus.fLiveWindow);
}
```

C:\VIDDCODE\VIDVW.CPP

```
void CVidecodeView::OnUpdateCapPreview(CCmdUI* pCmdUI)
{
    m_wndCap.GetStatus(m_CapStatus);
    pCmdUI->SetCheck (m_CapStatus.fLiveWindow);
    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized);
}

void CVidecodeView::OnUpdateCapOverlay(CCmdUI* pCmdUI)
{
    m_wndCap.GetStatus(m_CapStatus);
    pCmdUI->SetCheck (m_CapStatus.fOverlayWindow);
    pCmdUI->Enable (m_CapDriverCaps.fHasOverlay && m_CapDriverCaps.fCaptureInitialized);
}

// Raise the preview rate if we get focus
void CVidecodeView::OnSetFocus(CWnd* pOldWnd)
{
    CView::OnSetFocus(pOldWnd);

    ::SendMessage(m_wndCap.m_hWnd, WM_QUERYNEWPALETTE, 0, 0);
    m_wndCap.PreviewRate(66);
}

// Lower preview rate if we lose focus
void CVidecodeView::OnKillFocus(CWnd* pNewWnd)
{
    CView::OnKillFocus(pNewWnd);
    m_wndCap.PreviewRate(500);
}

////////////////////////////////////////////////
// Preview scaling
////////////////////////////////////////////////
void CVidecodeView::OnCapScale()
{
   RECT rc;

   GetClientRect (&rc);
   m_CapStatus.fScale = !m_CapStatus.fScale;
   m_wndCap.PreviewScale(m_CapStatus.fScale);  // Scale preview to the window
   if (m_CapStatus.fScale)
   {
      ::SetWindowPos(m_wndCap.m_hWnd, NULL, 0, 0, rc.right, rc.bottom, SWP_NOZORDER);
   }
   else
   {
      ::SetWindowPos (m_wndCap.m_hWnd, NULL, 0, 0,
          m_CapStatus.uiImageWidth, m_CapStatus.uiImageHeight, SWP_NOZORDER);
      ResizeNow();
   }
   GetParent()->InvalidateRect (NULL, FALSE);
}

void CVidecodeView::OnUpdateCapScale(CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized);
    pCmdUI->SetCheck (m_CapStatus.fScale);
}


////////////////////////////////////////////////
// Driver supplied dialogs
////////////////////////////////////////////////
```

C:\VIDDCODE\VIDVW.CPP

```
void CVidecodeView::OnCapDlgFormat()
{
     m_fDialogIsUp = TRUE;
     m_wndCap.DlgVideoFormat();      // Sets the format of captured video
     ResizeNow ();
     m_fDialogIsUp = FALSE;
}

void CVidecodeView::OnCapDlgDisplay()
{
     m_fDialogIsUp = TRUE;
     m_wndCap.DlgVideoDisplay();     // Controls appearance of video output
     m_fDialogIsUp = FALSE;
}

void CVidecodeView::OnCapDlgSource()
{
     m_fDialogIsUp = TRUE;
     m_wndCap.DlgVideoSource();      // Selects input channel, NTSC-PAL, etc.
     m_fDialogIsUp = FALSE;
}

void CVidecodeView::OnUpdateCapDlgDisplay(CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized && m_CapDriverCaps.fHasDlgVideoDisplay);
}

void CVidecodeView::OnUpdateCapDlgFormat(CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized &&
            m_CapDriverCaps.fHasDlgVideoFormat);
}

void CVidecodeView::OnUpdateCapDlgSource(CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized &&
            m_CapDriverCaps.fHasDlgVideoSource);
}

////////////////////////////////////////////////
// Palette creation
////////////////////////////////////////////////

void CVidecodeView::OnCapAutopal5()
{
     m_wndCap.PaletteAuto(5, 256);  // Sample 5 frames of 256 colors
}

void CVidecodeView::OnUpdateCapAutopal5(CCmdUI* pCmdUI)
{
    // Gray the capture palette menu item if capture device
    // doesn't support palettes

    pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized &&
            m_CapDriverCaps.fDriverSuppliesPalettes);
}

////////////////////////////////////////////////
// Capture Setup Dialog
////////////////////////////////////////////////
void CVidecodeView::OnCapSetup()
{
    CCapParmsDlg CapParmsDlg;
```

C:\VIDDCODE\VIDVW.CPP

```
    CapParmsDlg.m_FrameRate =   (float)1000000. / (float) m_CaptureParms.dwRequestMicroSecPerFrame
    CapParmsDlg.m_EnableAudio =  m_CaptureParms.fCaptureAudio;
    CapParmsDlg.m_DosBuffers =   m_CaptureParms.fUsingDOSMemory;
    CapParmsDlg.m_VideoBuffers = (int) m_CaptureParms.wNumVideoRequested;
    CapParmsDlg.m_DisableSmartDrv = m_CaptureParms.fDisableWriteCache;
    CapParmsDlg.hwndCap = m_wndCap.m_hWnd;

    m_fDialogIsUp = TRUE;
    if (CapParmsDlg.DoModal() == IDOK) {
        m_CaptureParms.dwRequestMicroSecPerFrame = (DWORD) (1000000. / CapParmsDlg.m_FrameRate);
        m_CaptureParms.fCaptureAudio = CapParmsDlg.m_EnableAudio;
        m_CaptureParms.fUsingDOSMemory = CapParmsDlg.m_DosBuffers;
        m_CaptureParms.wNumVideoRequested = (WORD) CapParmsDlg.m_VideoBuffers;
        m_CaptureParms.fDisableWriteCache = CapParmsDlg.m_DisableSmartDrv;
    }
    m_fDialogIsUp = FALSE;
}


void CVidecodeView::OnUpdateCapSetup(CCmdUI* pCmdUI)
{
   pCmdUI->Enable (m_CapDriverCaps.fCaptureInitialized);
}


////////////////////////////////////////////////
// Capture!!!
////////////////////////////////////////////////
void CVidecodeView::OnCapSequence()
{
    CAPINFOCHUNK cinfo;
    char szCopyright[] = "Copyright 1996 Welch Allyn, Inc";

    // Let's add a Copyright chunk to the capture file!
    cinfo.fccInfoID = mmioFOURCC ('I','C','O','P');
    cinfo.lpData = &szCopyright;
    cinfo.cbData = lstrlen (szCopyright) + 1;  // Add one for the NULL!
//    m_wndCap.FileSetInfoChunk(cinfo);

    // Inform the capture window of the capture settings
    m_wndCap.CaptureSetSetup(m_CaptureParms);

    // And finally, the point of it all...
    m_wndCap.CaptureSequence();
}

void CVidecodeView::OnUpdateCapSequence(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_CapDriverCaps.fCaptureInitialized);
}

////////////////////////////////////////////////
// File Menu functions
////////////////////////////////////////////////
void CVidecodeView::OnFileOpen()
{
    char szPath [_MAX_PATH];

    m_wndCap.FileGetCaptureFile(szPath, sizeof (szPath));

    CFileDialog dlgFile (TRUE /*bOpenFileDialog*/,
                "AVI" /*lpszDefExt*/,
                szPath /*lpszFileName*/,
```

C:\VIDDCODE\VIDVW.CPP

```
                OFN_HIDEREADONLY /* flags */,
                "(VFW) *.avi |*.avi||",
                NULL /* hwndParent */);

    dlgFile.m_ofn.lpstrFile = (LPSTR)szPath; // just fill in my own buffer

    if (dlgFile.DoModal() != IDOK)
        return;

    m_wndCap.FileSetCaptureFile((LPSTR)szPath);
}

////////////////////////////////////////////
// Copy only the portion of the capture file
// containing "real" data to another file
////////////////////////////////////////////
void CVidecodeView::OnFileSaveAs()
{
    char szPath [_MAX_PATH];

    lstrcpy (szPath, "*.avi");

    CFileDialog dlgFile (FALSE /*bOpenFileDialog*/,
                "AVI" /*lpszDefExt*/,
                szPath /*lpszFileName*/,
                OFN_HIDEREADONLY /* flags */,
                "(VFW) *.avi |*.avi||",
                NULL /* hwndParent */);

    dlgFile.m_ofn.lpstrFile = (LPSTR)szPath; // just fill in my own buffer

    if (dlgFile.DoModal() != IDOK)
        return;

    m_wndCap.FileSaveAs((LPSTR)szPath);
}

////////////////////////////////////////////
// Edit Menu functions
////////////////////////////////////////////
void CVidecodeView::OnEditPaste()
{
    m_wndCap.PalettePaste();
}

void CVidecodeView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_CapDriverCaps.fCaptureInitialized);
}

void CVidecodeView::OnEditCopy()
{
    m_wndCap.EditCopy();
}

void CVidecodeView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_CapDriverCaps.fCaptureInitialized);
}

void CVidecodeView::ResizeNow (void)
{
    RECT rc;
```

C:\VIDDCODE\VIDVW.CPP

```
    // Resize our window to the size of the captured video
    m_wndCap.GetStatus(m_CapStatus);
    SetRect (&rc, 0, 0, m_CapStatus.uiImageWidth, m_CapStatus.uiImageHeight);

    GetParent()-> CalcWindowRect (&rc);
    GetParent()-> SetWindowPos (NULL, 0, 0, rc.right - rc.left,
                rc.bottom - rc.top, SWP_NOZORDER | SWP_NOMOVE);
    GetParent()-> InvalidateRect (NULL, FALSE);
}

void CVidecodeView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    m_wndCap.GetStatus(m_CapStatus);

    if (m_CapStatus.fScale)
       ::SetWindowPos(m_wndCap.m_hWnd, NULL, 0, 0, cx, cy, SWP_NOZORDER);
    else
       ::SetWindowPos(m_wndCap.m_hWnd, NULL, 0, 0,
                m_CapStatus.uiImageWidth, m_CapStatus.uiImageHeight, SWP_NOZORDER);

    InvalidateRect (NULL, TRUE);

}
////////////////////////////////////////////////
// Drivers typically crash if they are displaying
// a dialog at end session.  For this reason,
// don't allow end session until the user closes
// all dialogs.
////////////////////////////////////////////////
BOOL CVidecodeView::OnQueryEndSession()
{
    if (m_fDialogIsUp) {
       MessageBeep (MB_ICONHAND);
       AfxMessageBox ("Close dialogs before exiting Windows.", MB_OK, 0);
       BringWindowToTop();
       return FALSE;
    }
    else
       return TRUE;
}


////////////////////////////////////////////////
// Connect to a driver (0-9)
////////////////////////////////////////////////
void CVidecodeView::OnUpdateCapDrv(CCmdUI* pCmdUI)
{
   pCmdUI->SetCheck ((BOOL)((int)pCmdUI->m_nID == (m_nDriverIndex + ID_CAP_DRV0)));
}


void CVidecodeView::ConnectToDriver(int nIndex)
{
    if (m_wndCap.DriverConnect(nIndex)) {
       m_nDriverIndex = nIndex;
       m_wndCap.DriverGetCaps(m_CapDriverCaps);
       ResizeNow ();
    }
    else {
       m_nDriverIndex = -1;
```

C:\VIDDCODE\VIDVW.CPP

```
        _fmemset (&m_CapDriverCaps, 0, sizeof (m_CapDriverCaps));
        _fmemset (&m_CapStatus, 0, sizeof (m_CapStatus));
    }
}

void CVidecodeView::OnCapDrv0()
{
    ConnectToDriver(0);
}

void CVidecodeView::OnCapDrv1()
{
    ConnectToDriver(1);
}

void CVidecodeView::OnCapDrv2()
{
    ConnectToDriver(2);
}

void CVidecodeView::OnCapDrv3()
{
    ConnectToDriver(3);
}

void CVidecodeView::OnCapDrv4()
{
    ConnectToDriver(4);
}

void CVidecodeView::OnCapDrv5()
{
    ConnectToDriver(5);
}

void CVidecodeView::OnCapDrv6()
{
    ConnectToDriver(6);
}

void CVidecodeView::OnCapDrv7()
{
    ConnectToDriver(7);
}

void CVidecodeView::OnCapDrv8()
{
    ConnectToDriver(8);
}

void CVidecodeView::OnCapDrv9()
{
    ConnectToDriver(9);
}

//============================================
//  HotKey Decode Mode Handler
//============================================
void CVidecodeView::OnAccelDecode()
{
   CString* pstrMsg = new CString("");

   if(MfxGetApp()->GetProfileInt("Settings","Decode Hot Key",DEFAULT_DEC_KB))
      DecodeFrame(pstrMsg);
```

C:\VIDDCODE\VIDVW.CPP

```
   if(pstrMsg->GetLength() > 0)
      OutputMessage(pstrMsg);
   m_strOldMsg = *pstrMsg;

   delete pstrMsg;
}

//=============================================
//  Timer Decode Mode Handler
//=============================================
void CVidecodeView::OnTimer(UINT nIDEvent)
{
   if(m_fDialogIsUp) return;

   CString* pstrMsg = new CString("");

   if(MfxGetApp()->GetProfileInt("Settings","Decode Timer",DEFAULT_DEC_TIMER))
   {
      BOOL bSuccess = DecodeFrame(pstrMsg);
      if((*pstrMsg != m_strOldMsg) && bSuccess)
         OutputMessage(pstrMsg);
      m_strOldMsg = *pstrMsg;
      CView::OnTimer(nIDEvent);
   }
   else
   {
      KillTimer(m_TimerID);
      m_TimerID = 0;
   }

   delete   pstrMsg;
}

//=============================================
//  Mouse Decode Mode Handler
//=============================================
void CVidecodeView::OnLButtonDown(UINT nFlags, CPoint point)
{
   CString* pstrMsg = new CString("");

   if(MfxGetApp()->GetProfileInt("Settings","Decode Mouse",DEFAULT_DEC_MOUSE))
      DecodeFrame(pstrMsg);
   if(pstrMsg->GetLength() > 0)
      OutputMessage(pstrMsg);
   m_strOldMsg = *pstrMsg;

   delete pstrMsg;
   CView::OnLButtonDown(nFlags, point);
}

//=============================================================
// Decode Frame
//
// This method captures a frame and then calls the decoder.
// A CString is returned to the caller which contains the
// message (if any) that was decoded.
//=============================================================
int CVidecodeView::DecodeFrame(CString* pstrMsg)
{
   int   DecodeReturn = FALSE;

   if(m_nDriverIndex == -1)
      return DecodeReturn;
```

C:\VIDDCODE\VIDVW.CPP

```
#if 1
   m_wndCap.CaptureDIB(DEFAULT_CAP_BMP);
#else
// videoOpen();
// videoFrame();
// videoClose();
#endif

   TRY
   {
      CRect    rc;
      CFile    fImage(DEFAULT_CAP_BMP, CFile::modeRead);
      CFileStatus status;

      // Open File as a CDib.
      CDib* pDib = new CDib;
      CArchive ar(&fImage, CArchive::load);
      pDib->Serialize(ar);

      GetClientRect(&rc);
      ScreenToClient(&rc);    // convert to parent coords

      // Convert the message to gray scale.
      CDib* pGrayDib = pDib->CreateGrayDIB(FALSE, CSize(rc.Width(),rc.Height()));
      delete pDib;

      int   bpp = 8;

      CSize Size = pGrayDib->GetImageSize();
      HGLOBAL hData = GlobalAlloc(GHND,(ULONG)Size.cx * Size.cy);
      UCHAR huge* pData = (UCHAR huge *)GlobalLock(hData);
      pGrayDib->GetTopDownData(pData);

      if((Size.cx > 0) && (Size.cy > 0))
      {
         char* pBuff = pstrMsg->GetBuffer(MAX_MSG_LEN);
         DecodeReturn = WA_Decode(Size.cx, Size.cy, pData, bpp, pBuff);
         pstrMsg->ReleaseBuffer();
      }
      m_wndCap.GrabFrameNoStop();

      delete pGrayDib;
      GlobalUnlock(hData);
      GlobalFree(hData);

   } // end TRY
   CATCH( CFileException, e )
   {
      // Don't do ANYTHING if there was a file error,
      // or if no file exists.  Just patiently wait
      // for a valid file before going off to the decoder.
   }
   END_CATCH

   return   DecodeReturn;
}

//************************************************
// Outputs the decoded message according to user
// output preference setting.
//************************************************
void  CVidecodeView::OutputMessage(CString *pstrMsg)
{
   ::MessageBeep(MB_OK);
```

C:\VIDDCODE\VIDVW.CPP

```
   int nOut = MfxGetApp()->GetProfileInt("Settings","Output",DEFAULT_OUT_MODE);
   switch(nOut)
   {
      case IDC_OUT_MSG:
      {
         CMsgDlg  MsgDlg(*pstrMsg);
         m_fDialogIsUp = TRUE;
         MsgDlg.DoModal();
         m_fDialogIsUp = FALSE;
         break;
      }

      case IDC_OUT_CLIP:
      {
         HGLOBAL hClipMsg = GlobalAlloc(GHND,pstrMsg->GetLength());
         char* pClipMsg = (char*) GlobalLock(hClipMsg);
         strcpy(pClipMsg,pstrMsg->GetBuffer(MAX_MSG_LEN));
         if(!OpenClipboard())
         {
            MessageBox("Failed to open clipboard","Error", MB_ICONEXCLAMATION);
         }
         else
         {
            ::EmptyClipboard();
            ::SetClipboardData(CF_TEXT,hClipMsg);
            ::CloseClipboard();
         }
         GlobalUnlock(hClipMsg);
         break;
      }

      //----------------------------------------
      // Send output to an edit window control
      //----------------------------------------
      case  IDC_OUT_EDIT:
         m_fDialogIsUp = TRUE;
         AfxMessageBox(*pstrMsg);
         m_fDialogIsUp = FALSE;
         break;

      case IDC_OUT_FILE:
         CString  strFile;
         strFile = MfxGetApp()->GetProfileString("Settings","Output File",DEFAULT_OUT_FILE);
         TRY
         {
            CFile fOutput(strFile, CFile::modeWrite | CFile::modeCreate);
            fOutput.Write(pstrMsg->GetBuffer(MAX_MSG_LEN),pstrMsg->GetLength());
            fOutput.Close();
         }
         CATCH( CFileException, e )
         {
            TRACE1("File could not be opened %d\n", e->m_cause);
         }
         END_CATCH
         break;
   } // end switch

} // end method
```

C:\VIDDCODE\VIDVW.H

```
////////////////////////////////////////////////////////////////////////////
// Vidvw.h : interface of the CVidecodeView class
//
////////////////////////////////////////////////////////////////////////////
#include "viddoc.h"

class CVidecodeView : public CView
{
protected: // create from serialization only
   CVidecodeView();
   DECLARE_DYNCREATE(CVidecodeView)
   static   BOOL  s_bMenuSet;

// Attributes
public:
   CVidecodeDoc*  GetDocument();
   CCapWnd        m_wndCap;
   CMciWnd        m_wndMci;
   CAPDRIVERCAPS  m_CapDriverCaps;
   CAPSTATUS      m_CapStatus;
   CAPTUREPARMS   m_CaptureParms;
   int            m_nDriverIndex;
   BOOL           m_fDialogIsUp;
protected:
   int            m_TimerID;
   CString        m_strOldMsg;

// Operations
public:
   void           SetupTimer(UINT Period);
   void           ConnectToDriver(int nIndex);
   void           ResizeNow (void);
   void           Layout(void);
   int            DecodeFrame(CString *pstrMsg);
   void           OutputMessage(CString *pstrMsg);

// Implementation
public:
   void           GetDriverList();
   virtual        ~CVidecodeView();
   virtual void   OnDraw(CDC* pDC) {};
#ifdef _DEBUG
   virtual void   AssertValid() const;
   virtual void   Dump(CDumpContext& dc) const;
#endif


// Generated message map functions
protected:
   //{{AFX_MSG(CVidecodeView)
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
   afx_msg void OnSize(UINT nType, int cx, int cy);
   afx_msg void OnCapOverlay();
   afx_msg void OnCapPreview();
   afx_msg void OnCapDlgFormat();
   afx_msg void OnCapDlgDisplay();
   afx_msg void OnCapDlgSource();
   afx_msg void OnCapAutopal5();
   afx_msg void OnCapSequence();
   afx_msg void OnFileOpen();
   afx_msg void OnFileSaveAs();
   afx_msg void OnEditPaste();
   afx_msg void OnEditCopy();
   afx_msg void OnUpdateCapPreview(CCmdUI* pCmdUI);
```

C:\VIDDCODE\VIDVW.H

```
    afx_msg void OnUpdateCapOverlay(CCmdUI* pCmdUI);
    afx_msg void OnCapScale();
    afx_msg void OnUpdateCapScale(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCapDlgDisplay(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCapDlgFormat(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCapDlgSource(CCmdUI* pCmdUI);
    afx_msg void OnCapSetup();
    afx_msg void OnCapDrv0();
    afx_msg void OnUpdateCapDrv(CCmdUI* pCmdUI);
    afx_msg void OnCapDrv1();
    afx_msg void OnCapDrv2();
    afx_msg void OnCapDrv3();
    afx_msg void OnCapDrv4();
    afx_msg void OnCapDrv5();
    afx_msg void OnCapDrv6();
    afx_msg void OnCapDrv7();
    afx_msg void OnCapDrv8();
    afx_msg void OnCapDrv9();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnKillFocus(CWnd* pNewWnd);
    afx_msg void OnUpdateCapAutopal5(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCapSequence(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCapSetup(CCmdUI* pCmdUI);
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
    afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
    afx_msg BOOL OnQueryEndSession();
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg void OnAccelDecode();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG    // debug version in captevw.cpp
inline CVidecodeDoc* CVidecodeView::GetDocument()
   { return (CVidecodeDoc*) m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

C:\VIDDCODE\CPPRMDLG.CPP

```
/*****************************************************************************
 *
 *   CapParms.cpp: Controls a dialog for setting capture parameters
 *
 *   Microsoft Video for Windows Capture Class Sample Program
 *
 *****************************************************************************/

#include "stdafx.h"
#include "videcode.h"
#include "viddoc.h"
#include "vidvw.h"
#include "mainfrm.h"
#include "cpprmdlg.h"

CCapParmsDlg::CCapParmsDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CCapParmsDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCapParmsDlg)
    m_FrameRate = 0;
    m_EnableAudio = FALSE;
    m_DosBuffers = FALSE;
    m_VideoBuffers = 0;
    m_DisableSmartDrv = FALSE;
    //}}AFX_DATA_INIT
}

void CCapParmsDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCapParmsDlg)
    DDX_Control(pDX, IDC_AUDIOFORMATTEXT, m_AudioFormatText);
    DDX_Text(pDX, IDC_FRAME_RATE, m_FrameRate);
    DDV_MinMaxFloat(pDX, m_FrameRate, 1.7e-002, 100.);
    DDX_Check(pDX, IDC_AUDIO, m_EnableAudio);
    DDX_Check(pDX, IDC_DOS_BUFFERS, m_DosBuffers);
    DDX_Text(pDX, IDC_VIDEO_BUFFERS, m_VideoBuffers);
    DDV_MinMaxInt(pDX, m_VideoBuffers, 1, 1000);
    DDX_Check(pDX, IDC_SMARTDRV, m_DisableSmartDrv);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCapParmsDlg, CDialog)
    //{{AFX_MSG_MAP(CCapParmsDlg)
    ON_WM_CREATE()
    ON_BN_CLICKED(IDC_SETAUDIOFORMAT, OnClickedSetaudioformat)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CCapParmsDlg message handlers

int CCapParmsDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CDialog::OnCreate(lpCreateStruct) == -1)
        return -1;


    return 0;
}


BOOL CCapParmsDlg::OnInitDialog()
{
```

C:\VIDDCODE\CPPRMDLG.H

```
// Capparms.h : header file
//
//

/////////////////////////////////////////////////////////////////////////////
// CCapParms dialog

class CCapParmsDlg : public CDialog
{
// Construction
public:
    CCapParmsDlg(CWnd* pParent = NULL);    // standard constructor

    HWND    hwndCap;
    void CCapParmsDlg::SetWaveFormatText (HWND hwndCap, CStatic *pcStatic);

// Dialog Data
    //{{AFX_DATA(CCapParmsDlg)
   enum { IDD = IDD_SETUP_DIALOG };
   CStatic  m_AudioFormatText;
   float m_FrameRate;
   BOOL  m_EnableAudio;
   BOOL  m_DosBuffers;
   int      m_VideoBuffers;
   BOOL  m_DisableSmartDrv;
   //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

    // Generated message map functions
    //{{AFX_MSG(CCapParmsDlg)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
   virtual void OnOK();
   afx_msg void OnClickedSetaudioformat();
   virtual BOOL OnInitDialog();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

C:\VIDDCODE\CPPRMDLG.CPP

```
    CDialog::OnInitDialog();

    SetWaveFormatText (hwndCap, &m_AudioFormatText);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCapParmsDlg::OnOK()
{
    // TODO: Add extra validation here

    CDialog::OnOK();
}

void CCapParmsDlg::SetWaveFormatText (HWND hwndCap, CStatic *pcStatic)
{
    LPWAVEFORMATEX lpwfex;
    DWORD dwSize;
    ACMFORMATDETAILS acmfd;

    pcStatic->SetWindowText("");

    // Get the current audio format
    if (dwSize = capGetAudioFormatSize (hwndCap)) {
        if (lpwfex = (LPWAVEFORMATEX) GlobalAllocPtr(GHND, dwSize)) {
            if (capGetAudioFormat(hwndCap, lpwfex, (WORD)dwSize)) {

                _fmemset (&acmfd, 0, sizeof (ACMFORMATDETAILS));
                acmfd.cbStruct = sizeof (ACMFORMATDETAILS);
                acmfd.pwfx = lpwfex;
                if (lpwfex->wFormatTag == WAVE_FORMAT_PCM)
                    dwSize = sizeof (PCMWAVEFORMAT);
                else
                    dwSize = sizeof (WAVEFORMATEX) + lpwfex -> cbSize;

                acmfd.cbwfx = dwSize;
                acmfd.dwFormatTag = lpwfex->wFormatTag;
                acmFormatDetails (NULL, &acmfd, ACM_FORMATDETAILSF_FORMAT);

                pcStatic->SetWindowText(acmfd.szFormat);
            }
            GlobalFreePtr(lpwfex) ;
        }
    }
}


void CCapParmsDlg::OnClickedSetaudioformat()
{
    ACMFORMATCHOOSE cfmt;
    LPWAVEFORMATEX lpwfex;
    DWORD dwSize;

    // Ask the ACM what the largest wave format is.....
    acmMetrics(NULL,
        ACM_METRIC_MAX_SIZE_FORMAT,
        &dwSize);

    // Get the current audio format
    dwSize = max (dwSize, capGetAudioFormatSize (hwndCap));
    lpwfex = (LPWAVEFORMATEX) GlobalAllocPtr(GHND, dwSize) ;
    capGetAudioFormat(hwndCap, lpwfex, (WORD)dwSize) ;

    _fmemset (&cfmt, 0, sizeof (ACMFORMATCHOOSE));
```

C:\VIDDCODE\CPPRMDLG.CPP

```
    cfmt.cbStruct = sizeof (ACMFORMATCHOOSE);
    cfmt.fdwStyle =  ACMFORMATCHOOSE_STYLEF_INITTOWFXSTRUCT;
    // Only allow PCM and hardware compressed audio formats
    cfmt.fdwEnum =      ACM_FORMATENUMF_HARDWARE |
                            ACM_FORMATENUMF_INPUT;
    cfmt.hwndOwner = GetParent()->GetSafeHwnd();
    cfmt.pwfx =     lpwfex;
    cfmt.cbwfx =    dwSize;

    if (!acmFormatChoose(&cfmt)) {
        capSetAudioFormat(hwndCap, lpwfex, (WORD)dwSize) ;
        SetWaveFormatText (hwndCap, &m_AudioFormatText);
    }

    GlobalFreePtr(lpwfex) ;
}
```

Claims

1. A process for capturing and decoding barcode information in real time from a continuously displayed image video signal, comprising the steps of:

   aiming an imaging apparatus at a target of interest, said target having either optically readable and bar coded information contained thereupon;
   continually displaying a real time image of said target from said imaging apparatus on a computer video monitor;
   selectively capturing an instantaneous image into the memory of said computer;
   decoding said image if said bar-code readable information is contained on said real time image while maintaining said displayed image on said display; and
   outputting the decoded information.

2. A process as recited in Claim 1, wherein said selective capturing step is performed automatically after a predetermined interval.

3. A process as recited in Claim 1, wherein said selective capturing step is performed by input through input means.

4. A process as recited in claim 3, wherein said input means is a keyboard.

5. A process as recited in claim 4, wherein said input means is a mouse.

6. A process as recited in Claim 1, wherein said decoded information is output on said monitor adjacent said real-time image.

7. A process as recited in Claim 1, including the step of storing said captured image into computer memory if no bar-code information is present on said image.

8. A process as recited in claim 3, wherein said input means is a signal from one of at least one of an external, remote and host device.

9. A process as recited in claim 1, wherein said decoded information is output to a computer file.

10. A process as recited in claim 1, wherein said decoded information is output to a computer keyboard buffer.

11. A process as recited in claim 1, wherein said decoded information is output to a pop-up dialogue box.

12. A process as recited in claim 1, wherein said decoded information is output to an application's clipboard.

13. A process as recited in claim 1, including the step of integrating said process with an external software application and outputting said decoded information to said application.

14. A process as recited in claim 1, including the step of storing said captured image into computer memory.

15. A process as recited in claim 14, including the step of storing captured images in which at least some of said images include bar code information.

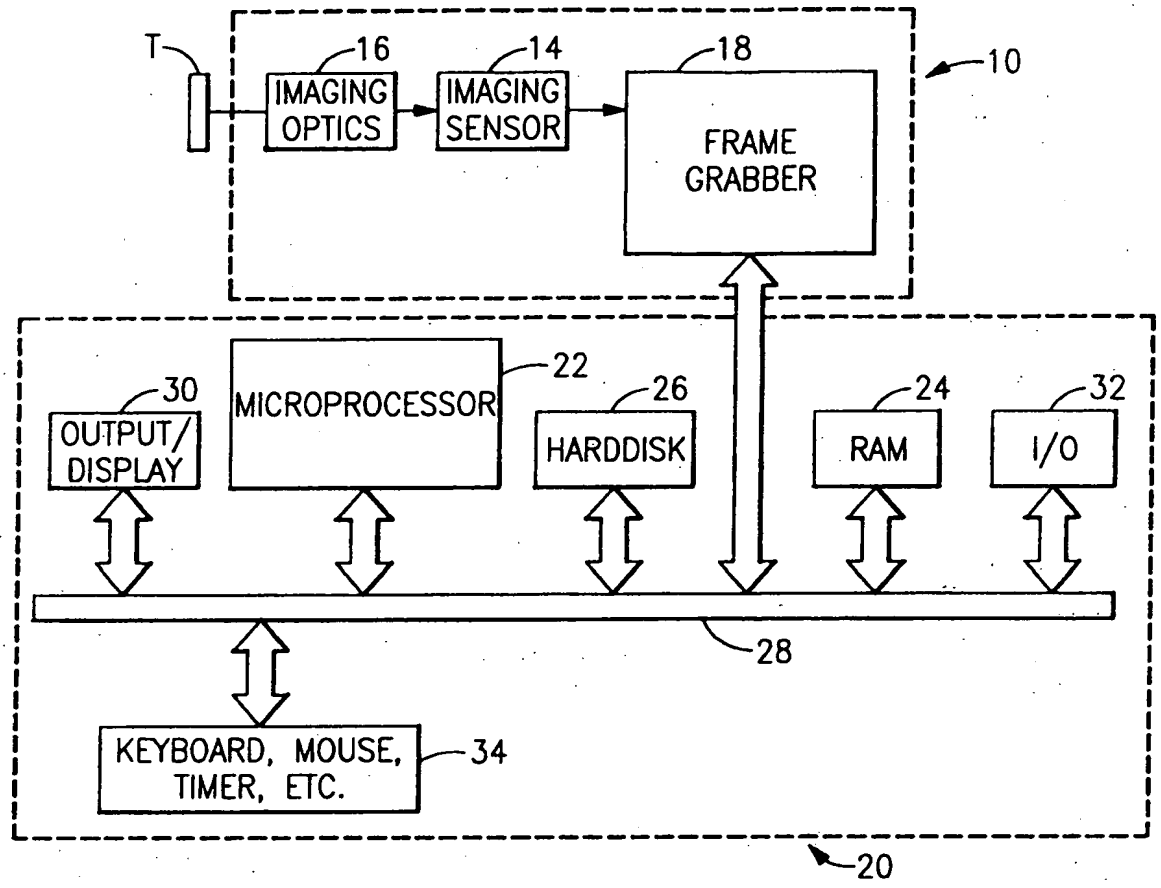

T
16
IMAGING OPTICS
14
IMAGING SENSOR
18
FRAME GRABBER
10
30
OUTPUT/ DISPLAY
MICROPROCESSOR
22
26
HARDDISK
24
RAM
32
I/O
28
KEYBOARD, MOUSE, TIMER, ETC.
34
20

FIG.1

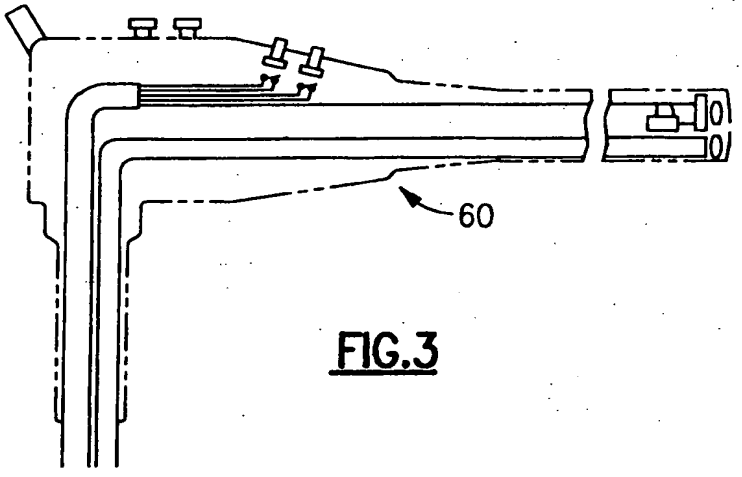

60

FIG.3

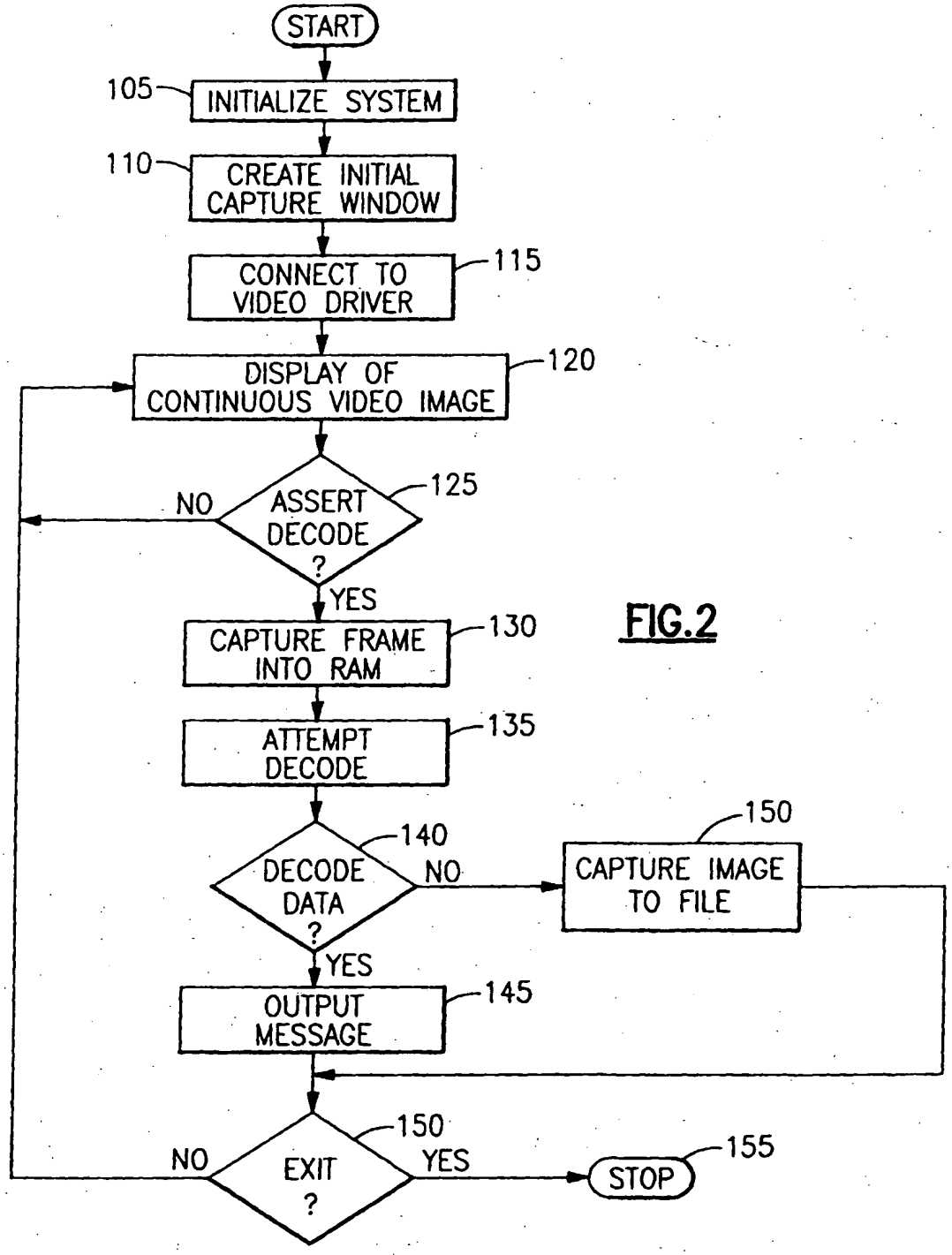

START
105
INITIALIZE SYSTEM
110
CREATE INITIAL CAPTURE WINDOW
115
CONNECT TO VIDEO DRIVER
120
DISPLAY OF CONTINUOUS VIDEO IMAGE
125
ASSERT DECODE ?
NO
YES
130
CAPTURE FRAME INTO RAM
135
ATTEMPT DECODE
140
DECODE DATA ?
NO
150
CAPTURE IMAGE TO FILE
YES
145
OUTPUT MESSAGE
150
EXIT ?
NO
YES
155
STOP

FIG.2

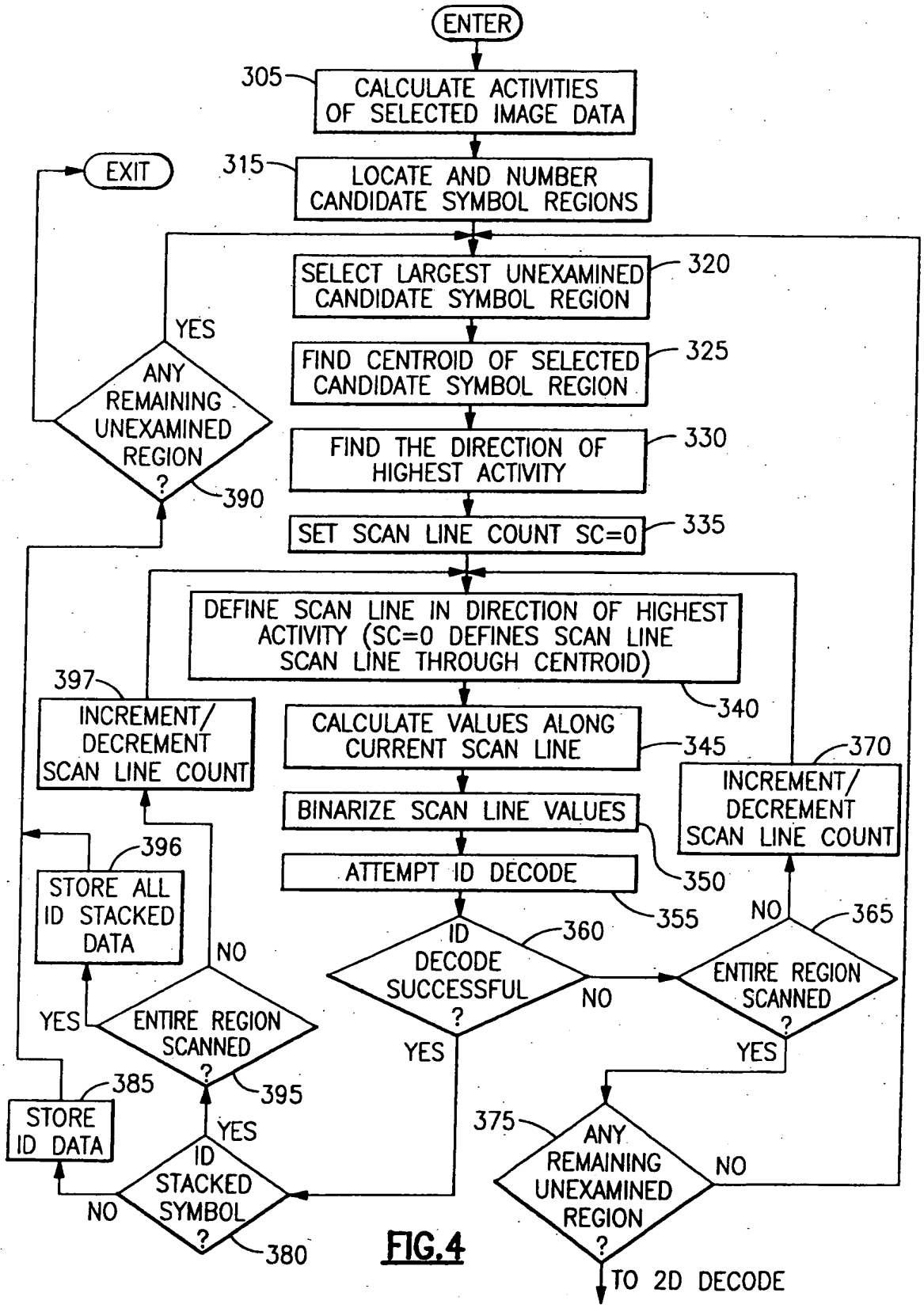

ENTER
305
CALCULATE ACTIVITIES OF SELECTED IMAGE DATA
315
LOCATE AND NUMBER CANDIDATE SYMBOL REGIONS
EXIT
SELECT LARGEST UNEXAMINED CANDIDATE SYMBOL REGION
320
FIND CENTROID OF SELECTED CANDIDATE SYMBOL REGION
325
FIND THE DIRECTION OF HIGHEST ACTIVITY
330
SET SCAN LINE COUNT SC=0
335
DEFINE SCAN LINE IN DIRECTION OF HIGHEST ACTIVITY (SC=0 DEFINES SCAN LINE SCAN LINE THROUGH CENTROID)
340
CALCULATE VALUES ALONG CURRENT SCAN LINE
345
BINARIZE SCAN LINE VALUES
350
ATTEMPT ID DECODE
355
ID DECODE SUCCESSFUL ?
360
YES
NO
ENTIRE REGION SCANNED ?
365
NO
YES
INCREMENT/ DECREMENT SCAN LINE COUNT
370
ANY REMAINING UNEXAMINED REGION ?
375
NO
YES
TO 2D DECODE
ID STACKED SYMBOL ?
380
NO
YES
STORE ID DATA
385
ENTIRE REGION SCANNED ?
395
NO
YES
STORE ALL ID STACKED DATA
396
INCREMENT/ DECREMENT SCAN LINE COUNT
397
ANY REMAINING UNEXAMINED REGION ?
390
YES

FIG.4

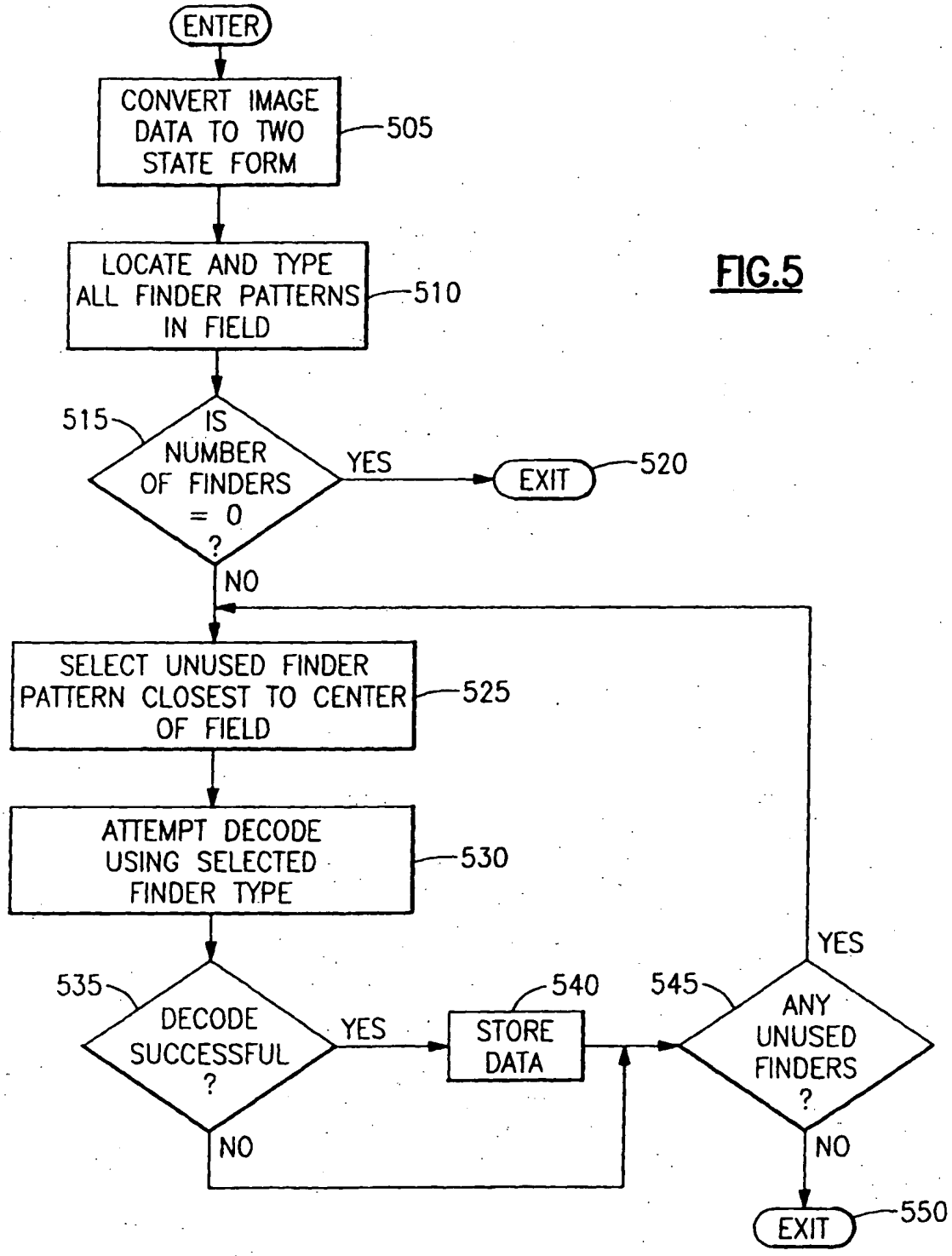

ENTER
CONVERT IMAGE DATA TO TWO STATE FORM
505
LOCATE AND TYPE ALL FINDER PATTERNS IN FIELD
510
515
IS NUMBER OF FINDERS = 0 ?
YES
EXIT
520
NO
SELECT UNUSED FINDER PATTERN CLOSEST TO CENTER OF FIELD
525
ATTEMPT DECODE USING SELECTED FINDER TYPE
530
535
DECODE SUCCESSFUL ?
YES
540
STORE DATA
545
ANY UNUSED FINDERS ?
YES
NO
NO
EXIT
550

FIG.5

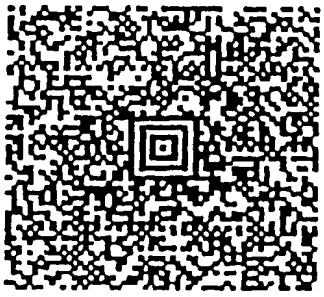


FIG.6

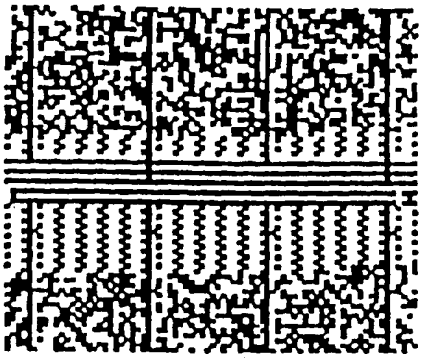


FIG.7

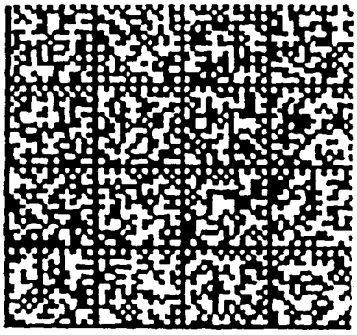


FIG.8

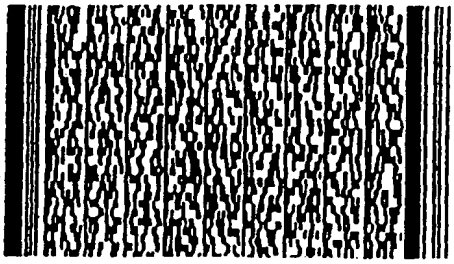
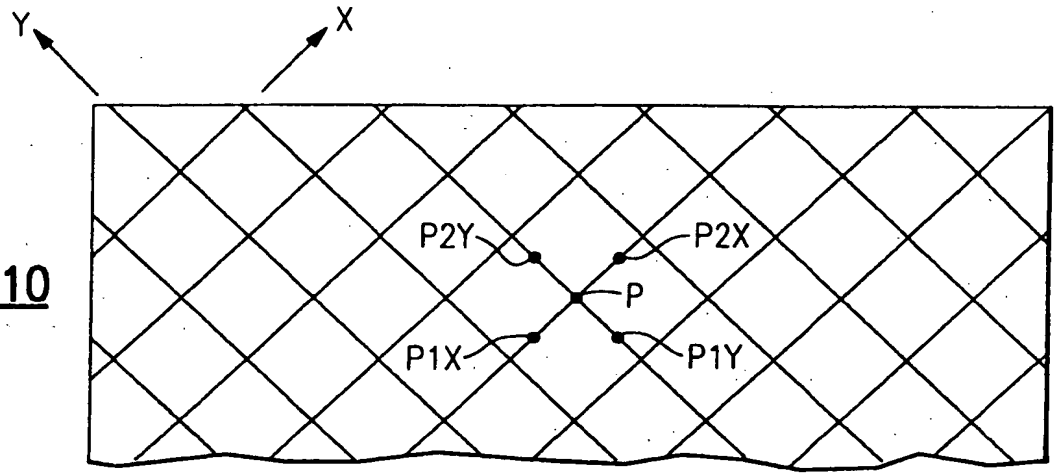


FIG.9

FIG.10



$$A_P = (P2X - P1X)^2 + (P2Y - P1Y)^2$$

FIG.11

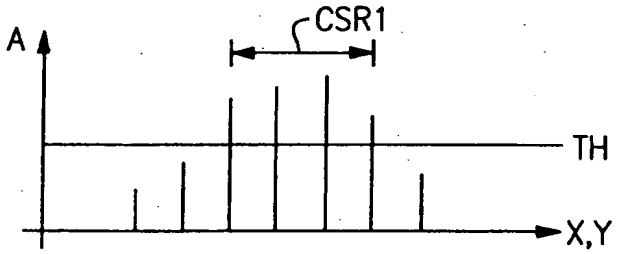


FIG.12

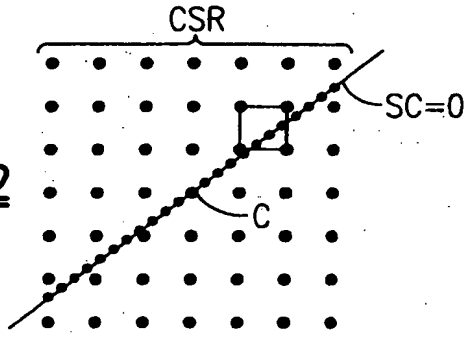


FIG.13

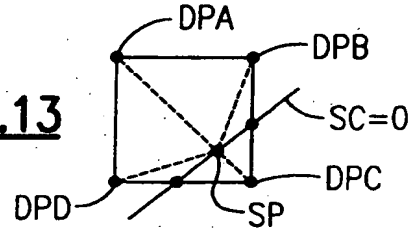


FIG.14